
nipype Documentation

Release 0.13.1

Neuroimaging in Python team

May 21, 2017, 03:41 PDT

1	User Guide	3
1.1	Download and install	3
1.2	Interface caching	4
1.3	Using Nipype Plugins	8
1.4	Configuration File	13
1.5	Debugging Nipype Workflows	15
1.6	Nipype Command Line Interface	16
1.7	DataGrabber and DataSink explained	17
1.8	The SelectFiles Interfaces	20
1.9	The Function Interface	21
1.10	MapNode, iterfield, and iterables explained	23
1.11	JoinNode, synchronize and itersource	27
1.12	Model Specification for First Level fMRI Analysis	32
1.13	Saving Workflows and Nodes to a file (experimental)	34
1.14	Using SPM with MATLAB Common Runtime	36
1.15	Using MIPAV, JIST, and CBS Tools	36
1.16	Running Nipype Interfaces from the command line (nipype_cmd)	36
1.17	Using Nipype with Amazon Web Services (AWS)	37
1.18	Resource Scheduling and Profiling with Nipype	38
1.19	Sphinx extensions	41
2	Changes in Nipype	45
2.1	0.13.1 (May 20, 2017)	45
2.2	0.13.0 (May 11, 2017)	45
2.3	0.13.0-rc1 (January 4, 2017)	46
2.4	Release 0.12.1 (August 3, 2016)	47
2.5	Release 0.12.0 (July 12, 2016)	47
2.6	Release 0.11.0 (September 15, 2015)	48
2.7	Release 0.10.0 (October 10, 2014)	50
2.8	Release 0.9.2 (January 31, 2014)	51
2.9	Release 0.9.1 (December 25, 2013)	51
2.10	Release 0.9.0 (December 20, 2013)	51
2.11	Release 0.8.0 (May 8, 2013)	52
2.12	Release 0.7.0 (Dec 18, 2012)	52
2.13	Release 0.6.0 (Jun 30, 2012)	53
2.14	Release 0.5.3 (Mar 23, 2012)	53

2.15	Release 0.5.2 (Mar 14, 2012)	53
2.16	Release 0.5 (Mar 10, 2012)	53
2.17	Release 0.4.1 (Jun 16, 2011)	54
2.18	Release 0.4 (Jun 11, 2011)	54
2.19	Release 0.3.4 (Jan 12, 2011)	55
2.20	Release 0.3.3 (Sep 16, 2010)	56
2.21	Release 0.3.2 (Aug 03, 2010)	56
2.22	Release 0.3.1 (Jul 29, 2010)	56
2.23	Release 0.3 (Jul 27, 2010)	56
3	API	59
3.1	caching.memory	59
3.2	interfaces.base	62
3.3	interfaces.matlab	144
3.4	pipeline.engine.base	154
3.5	pipeline.engine.nodes	155
3.6	pipeline.engine.utils	161
3.7	pipeline.engine.workflows	163
3.8	sphinxext.plot_workflow	166
4	Developer Guide	171
4.1	Interface Specifications	171
4.2	How to wrap a command line tool	178
4.3	How to wrap a MATLAB script	181
4.4	How to wrap a Python script	183
4.5	Working with <i>nipype</i> source code	184
4.6	Architecture (discussions from 2009)	195
4.7	W3C PROV support	198
4.8	Software using Nipype	198
4.9	Testing nipype	200
5	workflows.data	203
5.1	get_flirt_schedule()	203
6	workflows.dmri	205
6.1	workflows.dmri.camino.connectivity_mapping	205
6.2	workflows.dmri.camino.diffusion	206
6.3	workflows.dmri.camino.group_connectivity	207
6.4	workflows.dmri.connectivity.group_connectivity	208
6.5	workflows.dmri.connectivity.nx	211
6.6	workflows.dmri.dipy.denoise	213
6.7	workflows.dmri.fsl.artifacts	214
6.8	workflows.dmri.fsl.dti	231
6.9	workflows.dmri.fsl.epi	235
6.10	workflows.dmri.fsl.tbss	247
6.11	workflows.dmri.fsl.utils	259
6.12	workflows.dmri.mrtrix.connectivity_mapping	265
6.13	workflows.dmri.mrtrix.diffusion	266
6.14	workflows.dmri.mrtrix.group_connectivity	268
7	workflows.fmri	269
7.1	workflows.fmri.fsl.estimate	269
7.2	workflows.fmri.fsl.preprocess	273
7.3	workflows.fmri.spm.preprocess	282

8	workflows.misc	289
8.1	workflows.misc.utils	289
9	workflows.rsfmri	291
9.1	workflows.rsfmri.fsl.resting	291
10	workflows.smri	297
10.1	workflows.smri.ants.ANTSBuildTemplate	297
10.2	workflows.smri.ants.antsRegistrationBuildTemplate	299
10.3	workflows.smri.freesurfer.autorecon1	300
10.4	workflows.smri.freesurfer.autorecon2	300
10.5	workflows.smri.freesurfer.bem	300
10.6	workflows.smri.freesurfer.recon	302
10.7	workflows.smri.freesurfer.utils	304
10.8	workflows.smri.niftyreg.groupwise	310
11	dMRI: Camino, DTI	317
11.1	Introduction	317
12	dMRI: Connectivity - Camino, CMTK, FreeSurfer	323
12.1	Introduction	323
13	dMRI: Connectivity - MRtrix, CMTK, FreeSurfer	333
13.1	Introduction	333
13.2	Packages and Data Setup	333
13.3	Creating the workflow's nodes	335
13.4	Connecting the workflow	338
14	dMRI: DTI - Diffusion Toolkit, FSL	343
14.1	Setting up workflows	343
14.2	Data specific components	343
14.3	Setup for Diffusion Tensor Computation	344
14.4	Setup for Tracktography	345
14.5	Setup the pipeline that combines the two workflows: tractography and computeTensor	345
15	dMRI: HARDI - Diffusion Toolkit, FSL	347
15.1	Setting up workflows	347
15.2	Data specific components	347
15.3	Setup for ODF Computation	348
15.4	Setup for Tracktography	349
15.5	Setup the pipeline that combines the two workflows: tractography and compute_ODF	349
16	dMRI: DTI, FSL	351
16.1	Setting up workflows	351
16.2	Data specific components	351
16.3	Setup for Diffusion Tensor Computation	352
16.4	Setup for Tracktography	353
16.5	Setup the pipeline that combines the two workflows: tractography and computeTensor	354
17	dMRI: Group connectivity - Camino, FSL, FreeSurfer	357
17.1	Introduction	357
17.2	Main processing loop	358
18	dMRI: Group connectivity - MRtrix, FSL, FreeSurfer	361
18.1	Introduction	361
18.2	Main processing loop	362

19 dMRI: DTI - MRtrix, FSL	365
19.1 Introduction	365
20 dMRI: Preprocessing	371
20.1 Introduction	371
20.2 Setup for dMRI preprocessing	372
20.3 Connect nodes in workflow	373
21 dMRI: TBSS on NKI RS data	375
22 fMRI: OpenfMRI.org data, FSL, ANTS, c3daffine	379
23 fMRI: surface smooth - FreeSurfer, SPM	399
23.1 Preparing environment	399
23.2 Defining the workflow	399
24 fMRI: FSL	409
24.1 Preliminaries	409
24.2 Setting up workflows	409
24.3 Setup preprocessing workflow	410
24.4 Set up model fitting workflow	414
24.5 Set up fixed-effects workflow	414
24.6 Set up first-level workflow	415
24.7 Experiment specific components	415
24.8 Execute the pipeline	417
25 fMRI: FEEDS - FSL	419
25.1 Preliminaries	419
25.2 Experiment specific components	419
25.3 Execute the pipeline	421
26 fMRI: FSL reuse workflows	423
26.1 Preliminaries	423
26.2 Set up first-level workflow	424
26.3 Experiment specific components	425
26.4 Execute the pipeline	426
27 fMRI: NiPy GLM, SPM	429
27.1 Preliminaries	429
27.2 Preprocessing pipeline nodes	430
27.3 Set up analysis components	431
27.4 Setup the pipeline	432
28 fMRI: Coregistration - Slicer, BRAINS	435
28.1 Preliminaries	435
28.2 Preprocessing pipeline nodes	436
29 fMRI: SPM, FSL	439
29.1 Preliminaries	439
29.2 Preprocessing pipeline nodes	440
29.3 Set up analysis components	441
29.4 Setup the pipeline	442
29.5 Setup storage results	443
29.6 Setup level 2 pipeline	444
29.7 Execute the pipeline	444

30 fMRI: SPM Auditory dataset	447
30.1 Introduction	447
31 fMRI: DARTEL, SPM	453
31.1 Preliminaries	453
31.2 Setting up workflows	454
31.3 Setup preprocessing workflow	454
31.4 Set up analysis workflow	455
31.5 Preproc + Analysis pipeline	456
31.6 Data specific components	456
31.7 Experimental paradigm specific components	457
31.8 Setup the pipeline	458
31.9 Setup storage results	459
31.10 Execute the pipeline	460
31.11 Setup level 2 pipeline	460
31.12 Execute the second level pipeline	461
32 fMRI: Famous vs non-famous faces, SPM	463
32.1 Introduction	463
33 fMRI: SPM nested workflows	471
33.1 Preliminaries	471
33.2 Setting up workflows	472
33.3 Example of how to inline functions in connect()	472
33.4 Set-up preprocessing workflow	472
33.5 Set up analysis workflow	473
33.6 Preproc + Analysis pipeline	474
33.7 Data specific components	474
33.8 Experimental paradigm specific components	475
33.9 Setup the pipeline	476
33.10 Setup storage results	476
33.11 Execute the pipeline	477
33.12 Setup level 2 pipeline	477
33.13 Execute the second level pipeline	478
34 HOWTO: Using caching without using Workflow	479
35 rsfMRI: ANTS, FS, FSL, SPM, aCompCor	481
36 rsfMRI: ANTS, FS, FSL, NiPy, aCompCor	499
37 Paper: Smoothing comparison	517
38 sMRI: Using new ANTS for creating a T1 template	521
39 sMRI: Using ANTS for registration	525
40 sMRI: Using new ANTS for creating a T1 template (ITK4)	527
41 sMRI: USING CBS Tools for skullstripping	531
42 sMRI: FreeSurfer	533
43 sMRI: FSReconAll	535

44	sMRI: Regional Tessellation and Surface Smoothing	537
44.1	Introduction	537
44.2	Packages and Data Setup	537
44.3	Directories	538
44.4	Inputs	538
44.5	Outputs	538
44.6	Execution	538
45	Workflow from scratch	541
46	Workshop: Dartmouth College 2010	545
46.1	Using interfaces	546
46.2	Simple workflow	547
46.3	Another workflow	547
46.4	Reusing workflows	548
46.5	Visualizing workflows 1	548
46.6	Datasink	548
46.7	Datagrabber	548
46.8	Iterables	549
46.9	Visualizing workflows 2	550
47	algorithms.confounds	551
47.1	ACompCor	551
47.2	CompCor	552
47.3	ComputeDVARs	553
47.4	FramewiseDisplacement	554
47.5	NonSteadyStateDetector	555
47.6	TCompCor	555
47.7	TSNR	556
47.8	combine_mask_files()	557
47.9	compute_dvars()	557
47.10	compute_noise_components()	558
47.11	is_outlier()	558
47.12	plot_confound()	558
47.13	regress_poly()	558
48	algorithms.icc	559
48.1	ICC	559
48.2	ICC_rep_anova()	559
49	algorithms.mesh	561
49.1	ComputeMeshWarp	561
49.2	MeshWarpMaths	562
49.3	P2PDistance	562
49.4	TVTKBaseInterface	563
49.5	WarpPoints	563
50	algorithms.metrics	565
50.1	Distance	565
50.2	ErrorMap	566
50.3	FuzzyOverlap	566
50.4	Overlap	567
50.5	Similarity	568
51	algorithms.misc	571

51.1	AddCSVColumn	571
51.2	AddCSVRow	572
51.3	AddNoise	572
51.4	CalculateNormalizedMoments	573
51.5	CreateNifti	573
51.6	Distance	574
51.7	FuzzyOverlap	575
51.8	Gunzip	575
51.9	Matlab2CSV	576
51.10	MergeCSVFiles	576
51.11	MergeROIs	577
51.12	ModifyAffine	577
51.13	NormalizeProbabilityMapSet	578
51.14	Overlap	579
51.15	PickAtlas	579
51.16	SimpleThreshold	580
51.17	SplitROIs	580
51.18	TSNR	581
51.19	calc_moments()	582
51.20	makefmtlist()	582
51.21	maketypelist()	582
51.22	matlab2csv()	582
51.23	merge_csvs()	582
51.24	merge_rois()	582
51.25	normalize_tpms()	582
51.26	remove_identical_paths()	582
51.27	replaceext()	583
51.28	split_rois()	583
52	algorithms.modelgen	585
52.1	SpecifyModel	585
52.2	SpecifySPMModel	587
52.3	SpecifySparseModel	588
52.4	gcd()	590
52.5	gen_info()	590
52.6	orth()	590
52.7	scale_timings()	590
52.8	spm_hrf()	590
53	algorithms.rapidart	593
53.1	ArtifactDetect	593
53.2	StimulusCorrelation	595
54	interfaces.afni	597
54.1	interfaces.afni.preprocess	597
54.2	interfaces.afni.svm	638
54.3	interfaces.afni.utils	641
55	interfaces.ants	665
55.1	interfaces.ants.legacy	665
55.2	interfaces.ants.registration	670
55.3	interfaces.ants.resampling	678
55.4	interfaces.ants.segmentation	684
55.5	interfaces.ants.utils	707
55.6	interfaces.ants.visualization	712

56	interfaces.brainsuite	717
56.1	interfaces.brainsuite.brainsuite	717
57	interfaces.camino	747
57.1	interfaces.camino.calib	747
57.2	interfaces.camino.connectivity	751
57.3	interfaces.camino.convert	753
57.4	interfaces.camino.dti	765
57.5	interfaces.camino.odf	801
57.6	interfaces.camino.utils	808
58	interfaces.camino2trackvis	811
58.1	interfaces.camino2trackvis.convert	811
59	interfaces.cmtk	815
59.1	interfaces.cmtk.cmtk	815
59.2	interfaces.cmtk.convert	819
59.3	interfaces.cmtk.nbs	821
59.4	interfaces.cmtk.nx	822
59.5	interfaces.cmtk.parcellation	825
60	interfaces.diffusion_toolkit	829
60.1	interfaces.diffusion_toolkit.dti	829
60.2	interfaces.diffusion_toolkit.odf	832
60.3	interfaces.diffusion_toolkit.postproc	837
61	interfaces.dipy	841
61.1	interfaces.dipy.anisotropic_power	841
61.2	interfaces.dipy.base	842
61.3	interfaces.dipy.preprocess	842
61.4	interfaces.dipy.reconstruction	844
61.5	interfaces.dipy.simulate	847
61.6	interfaces.dipy.tensors	848
61.7	interfaces.dipy.tracks	850
62	interfaces.elastix	853
62.1	interfaces.elastix.registration	853
62.2	interfaces.elastix.utils	857
63	interfaces.freesurfer	859
63.1	interfaces.freesurfer.longitudinal	859
63.2	interfaces.freesurfer.model	862
63.3	interfaces.freesurfer.preprocess	893
63.4	interfaces.freesurfer.registration	932
63.5	interfaces.freesurfer.utils	938
64	interfaces.fsl	989
64.1	interfaces.fsl.ICA_AROMA	989
64.2	interfaces.fsl.dti	991
64.3	interfaces.fsl.epi	1011
64.4	interfaces.fsl.fix	1025
64.5	interfaces.fsl.maths	1029
64.6	interfaces.fsl.model	1049
64.7	interfaces.fsl.possum	1072
64.8	interfaces.fsl.preprocess	1074

64.9	interfaces.fsl.utils	1102
65	interfaces.minc	1137
65.1	interfaces.minc.base	1137
65.2	interfaces.minc.minc	1137
65.3	interfaces.minc.testdata	1186
66	interfaces.mipav	1187
66.1	interfaces.mipav.developer	1187
67	interfaces.mne	1215
67.1	interfaces.mne.base	1215
68	interfaces.mrtrix	1217
68.1	interfaces.mrtrix.convert	1217
68.2	interfaces.mrtrix.preprocess	1218
68.3	interfaces.mrtrix.tensors	1230
68.4	interfaces.mrtrix.tracking	1238
69	interfaces.mrtrix3	1253
69.1	interfaces.mrtrix3.base	1253
69.2	interfaces.mrtrix3.connectivity	1253
69.3	interfaces.mrtrix3.preprocess	1256
69.4	interfaces.mrtrix3.reconst	1258
69.5	interfaces.mrtrix3.tracking	1262
69.6	interfaces.mrtrix3.utils	1266
70	interfaces.niftyfit	1275
70.1	interfaces.niftyfit.asl	1275
70.2	interfaces.niftyfit.base	1278
70.3	interfaces.niftyfit.dwi	1278
70.4	interfaces.niftyfit.qt1	1285
71	interfaces.niftyreg	1289
71.1	interfaces.niftyreg.base	1289
71.2	interfaces.niftyreg.reg	1290
71.3	interfaces.niftyreg.regutils	1296
72	interfaces.niftyseg	1307
72.1	interfaces.niftyseg.base	1307
72.2	interfaces.niftyseg.em	1307
72.3	interfaces.niftyseg.label_fusion	1309
72.4	interfaces.niftyseg.lesions	1312
72.5	interfaces.niftyseg.maths	1314
72.6	interfaces.niftyseg.patchmatch	1323
72.7	interfaces.niftyseg.stats	1325
73	interfaces.nipy	1331
73.1	interfaces.nipy.model	1331
73.2	interfaces.nipy.preprocess	1333
73.3	interfaces.nipy.utils	1336
74	interfaces.nitime	1339
74.1	interfaces.nitime.analysis	1339

75	interfaces.semtools	1341
75.1	interfaces.semtools.brains.classify	1341
75.2	interfaces.semtools.brains.segmentation	1342
75.3	interfaces.semtools.brains.utilities	1345
75.4	interfaces.semtools.converters	1348
75.5	interfaces.semtools.diffusion.diffusion	1350
75.6	interfaces.semtools.diffusion.gtract	1359
75.7	interfaces.semtools.diffusion.maxcurvature	1390
75.8	interfaces.semtools.diffusion.tractography.commandlineonly	1391
75.9	interfaces.semtools.diffusion.tractography.fiberprocess	1392
75.10	interfaces.semtools.diffusion.tractography.fibertrack	1394
75.11	interfaces.semtools.diffusion.tractography.ukftractography	1396
75.12	interfaces.semtools.featurecreator	1399
75.13	interfaces.semtools.filtering.denoising	1400
75.14	interfaces.semtools.filtering.featuredetection	1402
75.15	interfaces.semtools.legacy.registration	1417
75.16	interfaces.semtools.registration.brainsfit	1418
75.17	interfaces.semtools.registration.brainsresample	1426
75.18	interfaces.semtools.registration.brainsresize	1428
75.19	interfaces.semtools.registration.specialized	1429
75.20	interfaces.semtools.segmentation.specialized	1437
75.21	interfaces.semtools.utilities.brains	1452
76	interfaces.slicer	1475
76.1	interfaces.slicer.base	1475
76.2	interfaces.slicer.converters	1475
76.3	interfaces.slicer.diffusion.diffusion	1478
76.4	interfaces.slicer.filtering.arithmetic	1489
76.5	interfaces.slicer.filtering.checkerboardfilter	1494
76.6	interfaces.slicer.filtering.denoising	1495
76.7	interfaces.slicer.filtering.extractskelton	1499
76.8	interfaces.slicer.filtering.histogrammatching	1500
76.9	interfaces.slicer.filtering.imagelabelcombine	1501
76.10	interfaces.slicer.filtering.morphology	1502
76.11	interfaces.slicer.filtering.n4itkbiasfieldcorrection	1504
76.12	interfaces.slicer.filtering.resamplescalarvectordwivolume	1506
76.13	interfaces.slicer.filtering.thresholdscalarvolume	1508
76.14	interfaces.slicer.filtering.votingbinaryholefillingimagefilter	1510
76.15	interfaces.slicer.legacy.converters	1511
76.16	interfaces.slicer.legacy.diffusion.denoising	1512
76.17	interfaces.slicer.legacy.filtering	1513
76.18	interfaces.slicer.legacy.registration	1515
76.19	interfaces.slicer.legacy.segmentation	1527
76.20	interfaces.slicer.quantification.changequantification	1528
76.21	interfaces.slicer.quantification.petstandarduptakevaluecomputation	1529
76.22	interfaces.slicer.registration.brainsfit	1530
76.23	interfaces.slicer.registration.brainsresample	1537
76.24	interfaces.slicer.registration.specialized	1539
76.25	interfaces.slicer.segmentation.simpleregiongrowingsegmentation	1548
76.26	interfaces.slicer.segmentation.specialized	1549
76.27	interfaces.slicer.surface	1554
76.28	interfaces.slicer.utilities	1562
77	interfaces.spm	1563

77.1	interfaces.spm.model	1563
77.2	interfaces.spm.preprocess	1575
77.3	interfaces.spm.utils	1593
78	interfaces.utility	1601
78.1	interfaces.utility.base	1601
78.2	interfaces.utility.csv	1605
78.3	interfaces.utility.wrappers	1606
79	interfaces.vista	1607
79.1	interfaces.vista.vista	1607
80	interfaces.bru2nii	1609
80.1	Bru2	1609
81	interfaces.c3	1611
81.1	C3dAffineTool	1611
82	interfaces.dcm2nii	1613
82.1	Dcm2nii	1613
82.2	Dcm2niix	1615
83	interfaces.dcmstack	1617
83.1	CopyMeta	1617
83.2	DcmStack	1617
83.3	GroupAndStack	1618
83.4	LookupMeta	1619
83.5	MergeNifti	1619
83.6	NiftiGeneratorBase	1620
83.7	SplitNifti	1620
83.8	make_key_func()	1620
83.9	sanitize_path_comp()	1621
84	interfaces.dynamic_slicer	1623
84.1	SlicerCommandLine	1623
85	interfaces.io	1625
85.1	DataFinder	1625
85.2	DataGrabber	1626
85.3	DataSink	1627
85.4	FreeSurferSource	1629
85.5	IOBase	1631
85.6	JSONFileGrabber	1631
85.7	JSONFileSink	1632
85.8	MySQLSink	1632
85.9	S3DataGrabber	1633
85.10	SQLiteSink	1634
85.11	SSHDataGrabber	1635
85.12	SelectFiles	1636
85.13	XNATSink	1637
85.14	XNATSource	1638
85.15	add_traits()	1639
85.16	copytree()	1639
85.17	push_file()	1639
85.18	quote_id()	1639

85.19	<code>unquote_id()</code>	1639
86	<code>interfaces.meshfix</code>	1641
86.1	<code>MeshFix</code>	1641
87	<code>interfaces.nilearn</code>	1645
87.1	<code>SignalExtraction</code>	1645
88	<code>interfaces.petpvc</code>	1647
88.1	<code>PETPVC</code>	1647
89	<code>interfaces.vtkbase</code>	1651
89.1	<code>configure_input_data()</code>	1651
89.2	<code>vtk_old()</code>	1651
89.3	<code>vtk_output()</code>	1651
	Bibliography	1653
	Python Module Index	1655

Previous versions: [0.13.0](#) [0.12.1](#)

Michael Notter's Nipype guide

Be sure to read [Michael's excellent tutorials](#).

Guides

- [User](#)

Release 0.13.1
Date May 21, 2017, 03:41 PDT

Download and install

This page covers the necessary steps to install Nipype.

Using docker

You can follow the [Nipype tutorial](#)
or use this docker container: *docker pull nipype/nipype*
or if you want to build custom docker containers with specific versions of software see [Neurodocker](#)

Using conda

Installing nipype from the conda-forge channel can be achieved by:

```
conda install --channel conda-forge nipype
```

It is possible to list all of the versions of nipype available on your platform with:

```
conda search nipype --channel conda-forge
```

For more information, please see <https://github.com/conda-forge/nipype-feedstock>

Using Pypi

The installation process is similar to other Python packages.
If you already have a Python environment set up, you can do:

```
pip install nipype
```

If you want to install all the optional features of nipype, use the following command:

```
pip install nipype[all]
```

While *all* installs everything, one can also install select components as listed below:

```
'doc': ['Sphinx>=1.4', 'matplotlib', 'pydotplus'],
'tests': ['pytest-cov', 'codecov'],
'nipy': ['nitime', 'nilearn', 'dipy', 'nipy', 'matplotlib'],
'profiler': ['psutil'],
'duecredit': ['duecredit'],
'xvfbwrapper': ['xvfbwrapper'],
```

Debian and Ubuntu

Add the [NeuroDebian](#) repository and install the `python-nipype` package using `apt-get` or your favorite package manager.

Mac OS X

The easiest way to get nipype running on Mac OS X is to install [Miniconda](#) and follow the instructions above. If you have a non-conda environment you can install nipype by typing:

```
pip install nipype
```

Note that the above procedure may require availability of `gcc` on your system path to compile the traits package.

From source

The most recent release is found here: <https://github.com/nipy/nipype/releases/latest>.

The development version: [\[zip tar.gz\]](#)

For previous versions: [prior downloads](#)

If you downloaded the source distribution named something like `nipype-x.y.tar.gz`, then unpack the tarball, change into the `nipype-x.y` directory and install nipype using:

```
pip install .
```

Note: Depending on permissions you may need to use `sudo`.

Testing the install

The best way to test the install is checking nipype's version and then running the tests:

```
python -c "import nipype; print(nipype.__version__)"
python -c "import nipype; nipype.test()"
```

Interface Dependencies

Nipype provides wrappers around many neuroimaging tools and contains some algorithms. These tools will need to be installed for Nipype to run. You can create containers with different versions of these tools installed using [Neurodocker](#)

Installation for developers

Developers should start here.

Developers can also use this docker container: `docker pull nipype/nipype:master`

Interface caching

This section details the interface-caching mechanism, exposed in the `nipype.caching` module.

Interface caching: why and how

- Pipelines (also called *workflows*) specify processing by an execution graph. This is useful because it opens the door to dependency checking and enable *i)* to minimize recomputations, *ii)* to have the execution engine transparently deal with intermediate file manipulations. They however do not blend in well with arbitrary Python code, as they must rely on their own execution engine.
- Interfaces give fine control of the execution of each step with a thin wrapper on the underlying software. As a result that can easily be inserted in Python code. However, they force the user to specify explicit input and output file names and cannot do any caching. This is why nipyte exposes an intermediate mechanism, *caching* that provides transparent output file management and caching within imperative Python code rather than a workflow.

A big picture view: using the Memory object

nipyte caching relies on the `Memory` class: it creates an execution context that is bound to a disk cache:

```
>>> from nipyte.caching import Memory
>>> mem = Memory(base_dir='.')
```

Note that the caching directory is a subdirectory called `nipyte_mem` of the given `base_dir`. This is done to avoid polluting the base director.

In the corresponding execution context, nipyte interfaces can be turned into callables that can be used as functions using the `Memory.cache()` method. For instance if we want to run the `fslMerge` command on a set of files:

```
>>> from nipyte.interface import fsl
>>> fsl_merge = mem.cache(fsl.Merge)
```

Note that the `Memory.cache()` method takes interfaces **classes**, and not instances.

The resulting `fsl_merge` object can be applied as a function to parameters, that will form the inputs of the `merge` fsl commands. Those inputs are given as keyword arguments, bearing the same name as the name in the inputs specs of the interface. In IPython, you can also get the argument list by using the `fsl_merge?` synthax to inspect the docs:

```
In [10]: fsl_merge?
String Form:PipeFunc(nipyte.interfaces.fsl.utils.Merge, base_dir=/home/varoquau/
↳dev/nipyte/nipyte/caching/nipyte_mem)
Namespace: Interactive
File: /home/varoquau/dev/nipyte/nipyte/caching/memory.py
Definition: fsl_merge(self, **kwargs)
Docstring:
Use fslmerge to concatenate images

Inputs
-----

Mandatory:
dimension: dimension along which the file will be merged
in_files: None

Optional:
args: Additional parameters to the command
environ: Environment variables (default={})
ignore_exception: Print an error message instead of throwing an exception in case_
↳the interface fails to run (default=False)
merged_file: None
output_type: FSL output type
```

```
Outputs
-----
merged_file: None
Class Docstring:
...
```

Thus `fsl_merge` is applied to parameters as such:

```
>>> results = fsl_merge(dimension='t', in_files=['a.nii.gz', 'b.nii.gz'])
INFO:workflow:Executing node faa7888f5955c961e5c6aa70cbd5c807 in dir: /home/
↳ varoquau/dev/nipyne/nipyne/caching/nipyne_mem/nipyne-interfaces-fsl-utils-Merge/
↳ faa7888f5955c961e5c6aa70cbd5c807
INFO:workflow:Running: fslmerge -t /home/varoquau/dev/nipyne/nipyne/caching/
↳ nipyne_mem/nipyne-interfaces-fsl-utils-Merge/faa7888f5955c961e5c6aa70cbd5c807/a_
↳ merged.nii /home/varoquau/dev/nipyne/nipyne/caching/a.nii.gz /home/varoquau/dev/
↳ nipyne/nipyne/caching/b.nii.gz
```

The results are standard nipyne nodes results. In particular, they expose an `outputs` attribute that carries all the outputs of the process, as specified by the docs.

```
>>> results.outputs.merged_file
'/home/varoquau/dev/nipyne/nipyne/caching/nipyne_mem/nipyne-interfaces-fsl-utils-
↳ Merge/faa7888f5955c961e5c6aa70cbd5c807/a_merged.nii'
```

Finally, and most important, if the node is applied to the same input parameters, it is not computed, and the results are reloaded from the disk:

```
>>> results = fsl_merge(dimension='t', in_files=['a.nii.gz', 'b.nii.gz'])
INFO:workflow:Executing node faa7888f5955c961e5c6aa70cbd5c807 in dir: /home/
↳ varoquau/dev/nipyne/nipyne/caching/nipyne_mem/nipyne-interfaces-fsl-utils-Merge/
↳ faa7888f5955c961e5c6aa70cbd5c807
INFO:workflow:Collecting precomputed outputs
```

Once the `Memory` is set up and you are applying it to data, an important thing to keep in mind is that you are using up disk cache. It might be useful to clean it using the methods that `Memory` provides for this: `Memory.clear_previous_runs()`, `Memory.clear_runs_since()`.

Example

A full-blown example showing how to stage multiple operations can be found in the `caching_example.py` file.

Usage patterns: working efficiently with caching

The goal of the `caching` module is to enable writing plain Python code rather than workflows. Use it: instead of data grabber nodes, use for instance the `glob` module. To vary parameters, use `for` loops. To make reusable code, write Python functions.

One good rule of thumb to respect is to avoid the usage of explicit filenames apart from the outermost inputs and outputs of your processing. The reason being that the caching mechanism of `nipyne.caching` takes care of generating the unique hashes, ensuring that, when you vary parameters, files are not overridden by the output of different computations.

Debugging

If you need to inspect the running environment of the nodes, it may be useful to know where they were executed. With `nipyne.caching`, you do not control this location as it is encoded by hashes.

To find out where an operation has been persisted, simply look in its output variable:

```
out.runtime.cwd
```

Finally, the more you explore different parameters, the more you risk creating cached results that will never be reused. Keep in mind that it may be useful to flush the cache using `Memory.clear_previous_runs()` or `Memory.clear_runs_since()`.

API reference

The main class of the `nipy.caching` module is the `Memory` class:

class `nipy.caching.Memory` (*base_dir*)

Memory context to provide caching for interfaces

Parameters `base_dir: string` :

The directory name of the location for the caching

Methods

<code>cache</code> (<i>interface</i>)	Returns a callable that caches the output of an interface
<code>clear_previous_runs</code> (<i>[warn]</i>)	Remove all the cache that where not used in the latest run of the memory object: i.e.
<code>clear_previous_runs</code> (<i>[warn]</i>)	Remove all the cache that where not used in the latest run of the memory object: i.e.

cache (*interface*)

Returns a callable that caches the output of an interface

Parameters `interface: nipy interface` :

The nipy interface class to be wrapped and cached

Returns `pipe_func: a PipeFunc callable object` :

An object that can be used as a function to apply the interface to arguments. Inputs of the interface are given as keyword arguments, bearing the same name as the name in the inputs specs of the interface.

Examples

```
>>> from tempfile import mkdtemp
>>> mem = Memory(mkdtemp())
>>> from nipy.interfaces import fsl
```

Here we create a callable that can be used to apply an `fsl.Merge` interface to files

```
>>> fsl_merge = mem.cache(fsl.Merge)
```

Now we apply it to a list of files. We need to specify the list of input files and the dimension along which the files should be merged.

```
>>> results = fsl_merge(in_files=['a.nii', 'b.nii'],
...                      dimension='t')
```

We can retrieve the resulting file from the outputs: `>>> results.outputs.merged_file` # doctest: +SKIP
`'...'`

clear_previous_runs (*warn=True*)

Remove all the cache that where not used in the latest run of the memory object: i.e. since the corresponding Python object was created.

Parameters warn: boolean, optional :

If true, echoes warning messages for all directory removed

clear_runs_since (*day=None, month=None, year=None, warn=True*)

Remove all the cache that where not used since the given date

Parameters day, month, year: integers, optional :

The integers specifying the latest day (in localtime) that a node should have been accessed to be kept. If not given, the current date is used.

warn: boolean, optional :

If true, echoes warning messages for all directory removed

Also used are the PipeFunc, callables that are returned by the `Memory.cache()` decorator:

class `nipyre.caching.memory.PipeFunc` (*interface, base_dir, callback=None*)

Callable interface to `nipyre.interface` objects

Use this to wrap `nipyre.interface` object and call them specifying their input with keyword arguments:

```
fsl_merge = PipeFunc(fsl.Merge, base_dir='.')
out = fsl_merge(in_files=files, dimension='t')
```

Methods

`__call__`(**kwargs)

`__init__` (*interface, base_dir, callback=None*)

Parameters interface: a nipyre interface class :

The interface class to wrap

base_dir: a string :

The directory in which the computation will be stored

callback: a callable :

An optional callable called each time after the function is called.

Using Nipyre Plugins

The workflow engine supports a plugin architecture for workflow execution. The available plugins allow local and distributed execution of workflows and debugging. Each available plugin is described below.

Current plugins are available for Linear, Multiprocessing, IPython distributed processing platforms and for direct processing on SGE, PBS, HTCondor, LSF, OAR, and SLURM. We anticipate future plugins for the Soma workflow.

Note: The current distributed processing plugins rely on the availability of a shared filesystem across computational nodes.

A variety of config options can control how execution behaves in this distributed context. These are listed later on in this page.

All plugins can be executed with:

```
workflow.run(plugin=PLUGIN_NAME, plugin_args=ARGS_DICT)
```

Optional arguments:

```
status_callback : a function handle
max_jobs : maximum number of concurrent jobs
max_tries : number of times to try submitting a job
```

```
retry_timeout : amount of time to wait between tries
```

Note: Except for the `status_callback`, the remaining arguments only apply to the distributed plugins: MultiProc/IPython(X)/SGE/PBS/HTCondor/HTCondorDAGMan/LSF

For example:

Plugins

Debug

This plugin provides a simple mechanism to debug certain components of a workflow without executing any node.

Mandatory arguments:

```
callable : A function handle that receives as arguments a node and a graph
```

The function callable will be called for every node from a topological sort of the execution graph.

Linear

This plugin runs the workflow one node at a time in a single process locally. The order of the nodes is determined by a topological sort of the workflow:

```
workflow.run(plugin='Linear')
```

MultiProc

Uses the [Python](#) multiprocessing library to distribute jobs as new processes on a local system.

Optional arguments:

```
n_procs : Number of processes to launch in parallel, if not set number of processors/threads will be automatically detected
```

To distribute processing on a multicore machine, simply call:

```
workflow.run(plugin='MultiProc')
```

This will use all available CPUs. If on the other hand you would like to restrict the number of used resources (to say 2 CPUs), you can call:

```
workflow.run(plugin='MultiProc', plugin_args={'n_procs' : 2})
```

IPython

This plugin provides access to distributed computing using [IPython](#) parallel machinery.

Note: We provide backward compatibility with [IPython](#) versions earlier than 0.10.1 using the IPythonX plugin. This plugin will be deprecated as of version 0.13 of Nipyte.

Please read the [IPython](#) documentation to determine how to setup your cluster for distributed processing. This typically involves calling `ipcluster`.

Once the clients have been started, any pipeline executed with:

```
workflow.run(plugin='IPython')
```

SGE/PBS

In order to use nipyne with **SGE** or **PBS** you simply need to call:

```
workflow.run(plugin='SGE')
workflow.run(plugin='PBS')
```

Optional arguments:

```
template: custom template file to use
qsub_args: any other command line args to be passed to qsub.
max_jobname_len: (PBS only) maximum length of the job name. Default 15.
```

For example, the following snippet executes the workflow on myqueue with a custom template:

```
workflow.run(plugin='SGE',
             plugin_args=dict(template='mytemplate.sh', qsub_args='-q myqueue'))
```

In addition to overall workflow configuration, you can use node level configuration for PBS/SGE:

```
node.plugin_args = {'qsub_args': '-l nodes=1:ppn=3'}
```

this would apply only to the node and is useful in situations, where a particular node might use more resources than other nodes in a workflow.

Note: Setting the keyword *overwrite* would overwrite any global configuration with this local configuration:

```
node.plugin_args = {'qsub_args': '-l nodes=1:ppn=3', 'overwrite': True}
```

SGEGraph

SGEGraph is an execution plugin working with Sun Grid Engine that allows for submitting entire graph of dependent jobs at once. This way Nipyne does not need to run a monitoring process - SGE takes care of this. The use of *SGEGraph* is preferred over **SGE** since the latter adds unnecessary load on the submit machine.

Note: When rerunning unfinished workflows using SGEGraph you may decide not to submit jobs for Nodes that previously finished running. This can speed up execution, but new or modified inputs that would previously trigger a Node to rerun will be ignored. The following option turns on this functionality:

```
workflow.run(plugin='SGEGraph', plugin_args = {'dont_resubmit_completed_jobs': True})
```

LSF

Submitting via LSF is almost identical to SGE above except for the optional arguments field:

```
workflow.run(plugin='LSF')
```

Optional arguments:

```
template: custom template file to use
bsub_args: any other command line args to be passed to bsub.
```

SLURM

Submitting via SLURM is almost identical to SGE above except for the optional arguments field:

```
workflow.run(plugin='SLURM')
```


Optional arguments:

```
template: custom template file to use
sbatch_args: any other command line args to be passed to bsub.
jobid_re: regular expression for custom job submission id search
```

SLURMGraph

SLURMGraph is an execution plugin working with SLURM that allows for submitting entire graph of dependent jobs at once. This way Nipyte does not need to run a monitoring process - SLURM takes care of this. The use of *SLURMGraph* plugin is preferred over the vanilla *SLURM* plugin since the latter adds unnecessary load on the submit machine.

Note: When rerunning unfinished workflows using *SLURMGraph* you may decide not to submit jobs for Nodes that previously finished running. This can speed up execution, but new or modified inputs that would previously trigger a Node to rerun will be ignored. The following option turns on this functionality:

```
workflow.run(plugin='SLURMGraph', plugin_args = {'dont_resubmit_completed_jobs':  
↪ True})
```

HTCondor

DAGMan

With its *DAGMan* component *HTCondor* (previously Condor) allows for submitting entire graphs of dependent jobs at once (similar to *SGEGraph* and *SLURMGraph*). With the *CondorDAGMan* plug-in Nipyte can utilize this functionality to submit complete workflows directly and in a single step. Consequently, and in contrast to other plug-ins, workflow execution returns almost instantaneously – Nipyte is only used to generate the workflow graph, while job scheduling and dependency resolution are entirely managed by *HTCondor*.

Please note that although *DAGMan* supports specification of data dependencies as well as data provisioning on compute nodes this functionality is currently not supported by this plug-in. As with all other batch systems supported by Nipyte, only *HTCondor* pools with a shared file system can be used to process Nipyte workflows. Workflow execution with *HTCondor DAGMan* is done by calling:

```
workflow.run(plugin='CondorDAGMan')
```

Job execution behavior can be tweaked with the following optional plug-in arguments. The value of most arguments can be a literal string or a filename, where in the latter case the content of the file will be used as the argument value:

```
submit_template : submit spec template for individual jobs in a DAG (see  
                  CondorDAGManPlugin.default_submit_template for the default.  
initial_specs : additional submit specs that are prepended to any job's  
                submit file  
override_specs : additional submit specs that are appended to any job's  
                submit file  
wrapper_cmd : path to an executable that will be started instead of a node  
              script. This is useful for wrapper script that execute certain  
              functionality prior or after a node runs. If this option is  
              given the wrapper command is called with the respective Python  
              executable and the path to the node script as final arguments  
wrapper_args : optional additional arguments to a wrapper command  
dagman_args : arguments to be prepended to the job execution script in the  
              dagman call  
block : if True the plugin call will block until Condor has finished  
        processing the entire workflow (default: False)
```

Please see the [HTCondor documentation](#) for details on possible configuration options and command line arguments.

Using the `wrapper_cmd` argument it is possible to combine Nipyte workflow execution with checkpoint/migration functionality offered by, for example, [DMTCP](#). This is especially useful in the case of workflows with long running nodes, such as Freesurfer's recon-all pipeline, where Condor's job prioritization algorithm could lead to jobs being evicted from compute nodes in order to maximize overall throughput. With checkpoint/migration enabled such a job would be checkpointed prior eviction and resume work from the checkpointed state after being rescheduled – instead of restarting from scratch.

On a Debian system, executing a workflow with support for checkpoint/migration for all nodes could look like this:

```
# define common parameters
dmtcp_hdr = """
should_transfer_files = YES
when_to_transfer_output = ON_EXIT_OR_EVICT
kill_sig = 2
environment = DMTCP_TMPDIR=./;JALIB_STDERR_PATH=/dev/null;DMTCP_PREFIX_ID=
↪$(CLUSTER)_$(PROCESS)
"""
shim_args = "--log %(basename)s.shimlog --stdout %(basename)s.shimout --stderr
↪%(basename)s.shimerr"
# run workflow
workflow.run(
    plugin='CondorDAGMan',
    plugin_args=dict(initial_specs=dmtcp_hdr,
                     wrapper_cmd='/usr/lib/condor/shim_dmtcp',
                     wrapper_args=shim_args)
)
```

OAR

In order to use nipyte with [OAR](#) you simply need to call:

```
workflow.run(plugin='OAR')
```

Optional arguments:

```
template: custom template file to use
oar_args: any other command line args to be passed to qsub.
max_jobname_len: (PBS only) maximum length of the job name. Default 15.
```

For example, the following snippet executes the workflow on myqueue with a custom template:

```
workflow.run(plugin='oar',
             plugin_args=dict(template='mytemplate.sh', oarsub_args='-q myqueue'))
```

In addition to overall workflow configuration, you can use node level configuration for OAR:

```
node.plugin_args = {'overwrite': True, 'oarsub_args': '-l "nodes=1/cores=3"'}
```

this would apply only to the node and is useful in situations, where a particular node might use more resources than other nodes in a workflow. You need to set the 'overwrite' flag to bypass the general settings-template you defined for the other nodes.

qsub emulation

Note: This plug-in is deprecated and users should migrate to the more robust and more versatile CondorDAGMan plug-in.

Despite the differences between HTCondor and SGE-like batch systems the plugin usage (incl. supported arguments) is almost identical. The HTCondor plugin relies on a `qsub` emulation script for HTCondor, called `condor_qsub` that can be obtained from a [Git repository on git.debian.org](https://git.debian.org). This script is currently not shipped with a standard HTCondor distribution, but is included in the HTCondor package from <http://neuro.debian.net>. It is sufficient to download this script and install it in any location on a system that is included in the `PATH` configuration.

Running a workflow in a HTCondor pool is done by calling:

```
workflow.run(plugin='Condor')
```

The plugin supports a limited set of `qsub` arguments (`qsub_args`) that cover the most common use cases. The `condor_qsub` emulation script translates `qsub` arguments into the corresponding HTCondor terminology and handles the actual job submission. For details on supported options see the manpage of `condor_qsub`.

Optional arguments:

```
qsub_args: any other command line args to be passed to condor_qsub.
```

Configuration File

Some of the system wide options of Nipype can be configured using a configuration file. Nipype looks for the file in the local folder under the name `nipype.cfg` and in `~/.nipype/nipype.cfg` (in this order). If an option will not be specified a default value will be assumed. The file is divided into following sections:

Logging

workflow_level How detailed the logs regarding workflow should be (possible values: `INFO` and `DEBUG`; default value: `INFO`)

filemanip_level How detailed the logs regarding file operations (for example overwriting warning) should be (possible values: `INFO` and `DEBUG`; default value: `INFO`)

interface_level How detailed the logs regarding interface execution should be (possible values: `INFO` and `DEBUG`; default value: `INFO`)

log_to_file Indicates whether logging should also send the output to a file (possible values: `true` and `false`; default value: `false`)

log_directory Where to store logs. (string, default value: home directory)

log_size Size of a single log file. (integer, default value: 254000)

log_rotate How many rotation should the log file make. (integer, default value: 4)

Execution

plugin This defines which execution plugin to use. (possible values: `Linear`, `MultiProc`, `SGE`, `IPython`; default value: `Linear`)

stop_on_first_crash Should the workflow stop upon first node crashing or try to execute as many nodes as possible? (possible values: `true` and `false`; default value: `false`)

stop_on_first_rerun Should the workflow stop upon first node trying to recompute (by that we mean rerunning a node that has been run before - this can happen due changed inputs and/or `hash_method` since the last run). (possible values: `true` and `false`; default value: `false`)

hash_method Should the input files be checked for changes using their content (slow, but 100% accurate) or just their size and modification date (fast, but potentially prone to errors)? (possible values: `content` and `timestamp`; default value: `timestamp`)

- keep_inputs*** Ensures that all inputs that are created in the nodes working directory are kept after node execution (possible values: `true` and `false`; default value: `false`)
- single_thread_matlab*** Should all of the Matlab interfaces (including SPM) use only one thread? This is useful if you are parallelizing your workflow using MultiProc or IPython on a single multicore machine. (possible values: `true` and `false`; default value: `true`)
- display_variable*** What `DISPLAY` variable should all command line interfaces be run with. This is useful if you are using `xnest` or `Xvfb` and you would like to redirect all spawned windows to it. (possible values: any X server address; default value: not set)
- remove_unnecessary_outputs*** This will remove any interface outputs not needed by the workflow. If the required outputs from a node changes, rerunning the workflow will rerun the node. Outputs of leaf nodes (nodes whose outputs are not connected to any other nodes) will never be deleted independent of this parameter. (possible values: `true` and `false`; default value: `true`)
- try_hard_link_datasink*** When the DataSink is used to produce an organized output file outside of nipyne's internal cache structure, a file system hard link will be attempted first. A hard link allows multiple file paths to point to the same physical storage location on disk if the conditions allow. By referring to the same physical file on disk (instead of copying files byte-by-byte) we can avoid unnecessary data duplication. If hard links are not supported for the source or destination paths specified, then a standard byte-by-byte copy is used. (possible values: `true` and `false`; default value: `true`)
- use_relative_paths*** Should the paths stored in results (and used to look for inputs) be relative or absolute. Relative paths allow moving the whole working directory around but may cause problems with symlinks. (possible values: `true` and `false`; default value: `false`)
- local_hash_check*** Perform the hash check on the job submission machine. This option minimizes the number of jobs submitted to a cluster engine or a multiprocessing pool to only those that need to be rerun. (possible values: `true` and `false`; default value: `true`)
- job_finished_timeout*** When batch jobs are submitted through, SGE/PBS/Condor they could be killed externally. Nipyne checks to see if a results file exists to determine if the node has completed. This timeout determines for how long this check is done after a job finish is detected. (float in seconds; default value: 5)
- remove_node_directories (EXPERIMENTAL)*** Removes directories whose outputs have already been used up. Doesn't work with `IdentiInterface` or any node that patches data through (without copying) (possible values: `true` and `false`; default value: `false`)
- stop_on_unknown_version*** If this is set to `True`, an underlying interface will raise an error, when no version information is available. Please notify developers or submit a patch.
- parameterize_dirs*** If this is set to `True`, the node's output directory will contain full parameterization of any iterable, otherwise parameterizations over 32 characters will be replaced by their hash. (possible values: `true` and `false`; default value: `true`)
- poll_sleep_duration*** This controls how long the job submission loop will sleep between submitting all pending jobs and checking for job completion. To be nice to cluster schedulers the default is set to 2 seconds.
- xvfb_max_wait*** Maximum time (in seconds) to wait for `Xvfb` to start, if the `_redirect_x` parameter of an `Interface` is `True`.
- crashfile_format*** This option controls the file type of any crashfile generated. `Pklz` crashfiles allow interactive debugging and rerunning of nodes, while text crashfiles allow portability across machines and shorter load time. (possible values: `pklz` and `txt`; default value: `pklz`)

Example

```
[logging]
workflow_level = DEBUG

[execution]
stop_on_first_crash = true
hash_method = timestamp
display_variable = :1
```

Workflow.config property has a form of a nested dictionary reflecting the structure of the .cfg file.

```
myworkflow = pe.Workflow()
myworkflow.config['execution'] = {'stop_on_first_rerun': 'True',
                                  'hash_method': 'timestamp'}
```

You can also directly set global config options in your workflow script. An example is shown below. This needs to be called before you import the pipeline or the logger. Otherwise logging level will not be reset.

```
from nipyte import config
cfg = dict(logging=dict(workflow_level = 'DEBUG'),
           execution={'stop_on_first_crash': False,
                     'hash_method': 'content'})
config.update_config(cfg)
```

Enabling logging to file

By default, logging to file is disabled. One can enable and write the file to a location of choice as in the example below.

```
import os
from nipyte import config, logging
config.update_config({'logging': {'log_directory': os.getcwd(),
                                  'log_to_file': True}})
logging.update_logging(config)
```

The logging update line is necessary to change the behavior of logging such as output directory, logging level, etc.,.

Debug configuration

To enable debug mode, one can insert the following lines:

```
from nipyte import config, logging
config.enable_debug_mode()
logging.update_logging(config)
```

In this mode the following variables are set:

```
config.set('execution', 'stop_on_first_crash', 'true')
config.set('execution', 'remove_unnecessary_outputs', 'false')
config.set('logging', 'workflow_level', 'DEBUG')
config.set('logging', 'interface_level', 'DEBUG')
```

Debugging Nipyte Workflows

Throughout Nipyte we try to provide meaningful error messages. If you run into an error that does not have a meaningful error message please let us know so that we can improve error reporting.

Here are some notes that may help debugging workflows or understanding performance issues.

1. Always run your workflow first on a single iterable (e.g. subject) and gradually increase the execution distribution complexity (Linear->MultiProc-> SGE).
2. Use the debug config mode. This can be done by setting:

```
from nipyte import config
config.enable_debug_mode()
```

as the first import of your nipyte script. To enable debug logging use:

```
from nipyte import logging
logging.update_logging(config)
```

Note: Turning on debug will rerun your workflows and will rerun them after debugging is turned off.

3. There are several configuration options that can help with debugging. See [Configuration File](#) for more details:

```
keep_inputs
remove_unnecessary_outputs
stop_on_first_crash
stop_on_first_rerun
```

4. When running in distributed mode on cluster engines, it is possible for a node to fail without generating a crash file in the crashdump directory. In such cases, it will store a crash file in the *batch* directory.
5. All Nipyte crashfiles can be inspected with the *nipytecli crash* utility.
6. The *nipytecli search* command allows you to search for regular expressions in the tracebacks of the Nipyte crashfiles within a log folder.
7. Nipyte determines the hash of the input state of a node. If any input contains strings that represent files on the system path, the hash evaluation mechanism will determine the timestamp or content hash of each of those files. Thus any node with an input containing huge dictionaries (or lists) of file names can cause serious performance penalties.
8. For HUGE data processing, 'stop_on_first_crash': 'False', is needed to get the bulk of processing done, and then 'stop_on_first_crash': 'True', is needed for debugging and finding failing cases. Setting 'stop_on_first_crash': 'False' is a reasonable option when you would expect 90% of the data to execute properly.
9. Sometimes nipyte will hang as if nothing is going on and if you hit Ctrl+C you will get a *Concurrent-LogHandler* error. Simply remove the *pipeline.lock* file in your home directory and continue.
10. One many clusters with shared NFS mounts synchronization of files across clusters may not happen before the typical NFS cache timeouts. When using PBS/LSF/SGE/Condor plugins in such cases the workflow may crash because it cannot retrieve the node result. Setting the *job_finished_timeout* can help:

```
workflow.config['execution']['job_finished_timeout'] = 65
```

Nipyte Command Line Interface

The Nipyte Command Line Interface allows a variety of operations:

```
$ nipytecli
Usage: nipytecli [OPTIONS] COMMAND [ARGS]...

Options:
  -h, --help  Show this message and exit.

Commands:
  convert  Export nipyte interfaces to other formats.
  crash    Display Nipyte crash files.
  run      Run a Nipyte Interface.
  search   Search for tracebacks content.
  show     Print the content of Nipyte node .pklz file.
```

These have replaced previous nipyte command line tools such as *nipyte_display_crash*, *nipyte_crash_search*, *nipyte2boutiques*, *nipyte_cmd* and *nipyte_display_pklz*.

DataGrabber and DataSink explained

In this chapter we will try to explain the concepts behind DataGrabber and *DataSink*.

Why do we need these interfaces?

A typical workflow takes data as input and produces data as the result of one or more operations. One can set the data required by a workflow directly as illustrated below.

```
from fsl_tutorial2 import preproc
preproc.base_dir = os.path.abspath('.')
preproc.inputs.inputs.spec.func = os.path.abspath('data/s1/f3.nii')
preproc.inputs.inputs.spec.struct = os.path.abspath('data/s1/struct.nii')
preproc.run()
```

Typical neuroimaging studies require running workflows on multiple subjects or different parameterizations of algorithms. One simple approach to that would be to simply iterate over subjects.

```
from fsl_tutorial2 import preproc
for name in subjects:
    preproc.base_dir = os.path.abspath('.')
    preproc.inputs.inputs.spec.func = os.path.abspath('data/%s/f3.nii'%name)
    preproc.inputs.inputs.spec.struct = os.path.abspath('data/%s/struct.nii'%name)
    preproc.run()
```

However, in the context of complex workflows and given that users typically arrange their imaging and other data in a semantically hierarchical data store, an alternative mechanism for reading and writing the data generated by a workflow is often necessary. As the names suggest DataGrabber is used to get at data stored in a shared file system while *DataSink* is used to store the data generated by a workflow into a hierarchical structure on disk.

DataGrabber

DataGrabber is an interface for collecting files from hard drive. It is very flexible and supports almost any file organization of your data you can imagine.

You can use it as a trivial use case of getting a fixed file. By default, DataGrabber stores its outputs in a field called outfiles.

```
import nipy.interfaces.io as nio
datasource1 = nio.DataGrabber()
datasource1.inputs.base_directory = os.getcwd()
datasource1.inputs.template = 'data/s1/f3.nii'
datasource1.inputs.sort_filelist = True
results = datasource1.run()
```

Or you can get at all uncompressed NIFTI files starting with the letter 'f' in all directories starting with the letter 's'.

```
datasource2.inputs.base_directory = '/mass'
datasource2.inputs.template = 'data/s*/f*.nii'
datasource1.inputs.sort_filelist = True
```

Two special inputs were used in these previous cases. The input *base_directory* indicates in which directory to search, while the input *template* indicates the string template to match. So in the previous case datagrabber is looking for path matches of the form */mass/data/s*/f**.

Note: When used with wildcards (e.g., *s** and *f** above) DataGrabber does not return data in sorted order. In order to force it to return data in sorted order, one needs to set the input *sorted* = *True*. However, when explicitly specifying an order as we will see below, *sorted* should be set to *False*.

More useful cases arise when the template can be filled by other inputs. In the example below, we define an input field for *datagrabber* called *run*. This is then used to set the template (see %d in the template).

```
datasource3 = nio.DataGrabber(infields=['run'])
datasource3.inputs.base_directory = os.getcwd()
datasource3.inputs.template = 'data/s1/f%d.nii'
datasource1.inputs.sort_filelist = True
datasource3.inputs.run = [3, 7]
```

This will return files *basedir/data/s1/f3.nii* and *basedir/data/s1/f7.nii*. We can take this a step further and pair subjects with runs.

```
datasource4 = nio.DataGrabber(infields=['subject_id', 'run'])
datasource4.inputs.template = 'data/%s/f%d.nii'
datasource1.inputs.sort_filelist = True
datasource4.inputs.run = [3, 7]
datasource4.inputs.subject_id = ['s1', 's3']
```

This will return files *basedir/data/s1/f3.nii* and *basedir/data/s3/f7.nii*.

A more realistic use-case

In a typical study one often wants to grab different files for a given subject and store them in semantically meaningful outputs. In the following example, we wish to retrieve all the functional runs and the structural image for the subject 's1'.

```
datasource = nio.DataGrabber(infields=['subject_id'], outfields=['func', 'struct
↪'])
datasource.inputs.base_directory = 'data'
datasource.inputs.template = '*'
datasource1.inputs.sort_filelist = True
datasource.inputs.field_template = dict(func='%s/f%d.nii',
                                         struct='%s/struct.nii')
datasource.inputs.template_args = dict(func=[['subject_id', [3,5,7,10]]],
                                         struct=[['subject_id']])
datasource.inputs.subject_id = 's1'
```

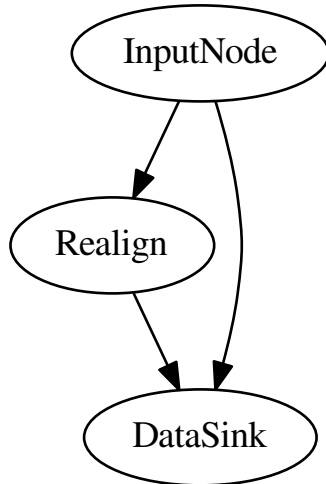
Two more fields are introduced: *field_template* and *template_args*. These fields are both dictionaries whose keys correspond to the *outfields* keyword. The *field_template* reflects the search path for each output field, while the *template_args* reflect the inputs that satisfy the template. The inputs can either be one of the named inputs specified by the *infields* keyword arg or it can be raw strings or integers corresponding to the template. For the *func* output, the %s in the *field_template* is satisfied by *subject_id* and the %d is field in by the list of numbers.

Note: We have not set *sorted* to *True* as we want the DataGrabber to return the functional files in the order it was specified rather than in an alphabetic sorted order.

DataSink

A workflow working directory is like a **cache**. It contains not only the outputs of various processing stages, it also contains various extraneous information such as execution reports, hashfiles determining the input state of processes. All of this is embedded in a hierarchical structure that reflects the iterables that have been used in the workflow. This makes navigating the working directory a not so pleasant experience. And typically the user is interested in preserving only a small percentage of these outputs. The *DataSink* interface can be used to extract components from this *cache* and store it at a different location. For XNAT-based storage, see *XNATSink* .

Note: Unlike other interfaces, a *DataSink*'s inputs are defined and created by using the workflow connect



MapNode

In order to use *DataSink* inside a MapNode, it's inputs have to be defined inside the constructor using the *infields* keyword arg.

Parameterization

As discussed in *MapNode, iterfield, and iterables explained*, one can run a workflow iterating over various inputs using the iterables attribute of nodes. This means that a given workflow can have multiple outputs depending on how many iterables are there. Iterables create working directory subfolders such as *_iterable_name_value*. The *parameterization* input parameter controls whether the data stored using *DataSink* is in a folder structure that contains this iterable information or not. It is generally recommended to set this to *True* when using multiple nested iterables.

Substitutions

The *substitutions* and *regexp_substitutions* inputs allow users to modify the output destination path and name of a file. Substitutions are a list of 2-tuples and are carried out in the order in which they were entered. Assuming that the output path of a file is:

```
/root/container/_variable_1/file_subject_realigned.nii
```

we can use substitutions to clean up the output path.

```
datsink.inputs.substitutions = [('_variable_', 'variable'),  
                                ('file_subject_', '')]
```

This will rewrite the file as:

```
/root/container/variable_1/realigned.nii
```

Note: In order to figure out which substitutions are needed it is often useful to run the workflow on a limited set of iterables and then determine the substitutions.

The SelectFiles Interfaces

Nipype 0.9 introduces a new interface for intelligently finding files on the disk and feeding them into your workflows: *SelectFiles*. SelectFiles is intended as a simpler alternative to the DataGrabber interface that was discussed previously in *DataGrabber and DataSink explained*.

SelectFiles is built on Python *format strings*, which are similar to the Python string interpolation feature you are likely already familiar with, but advantageous in several respects. Format strings allow you to replace named sections of template strings set off by curly braces (*{}*), possibly filtered through a set of functions that control how the values are rendered into the string. As a very basic example, we could write

```
msg = "This workflow uses {package}"
```

and then format it with keyword arguments:

```
print msg.format(package="FSL")
```

SelectFiles only requires that you provide templates that can be used to find your data; the actual formatting happens behind the scenes.

Consider a basic example in which you want to select a T1 image and multiple functional images for a number of subjects. Invoking SelectFiles in this case is quite straightforward:

```
from nipyne import SelectFiles
templates = dict(T1="data/{subject_id}/struct/T1.nii",
                 epi="data/{subject_id}/func/epi_run*.nii")
sf = SelectFiles(templates)
```

SelectFiles will take the *templates* dictionary and parse it to determine its own inputs and outputs. Specifically, each name used in the format spec (here just *subject_id*) will become an interface input, and each key in the dictionary (here *T1* and *epi*) will become interface outputs. The *templates* dictionary thus succinctly links the node inputs to the appropriate outputs. You'll also note that, as was the case with DataGrabber, you can use basic *glob* syntax to match multiple files for a given output field. Additionally, any of the conversions outlined in the Python documentation for format strings can be used in the templates.

There are a few other options that help make SelectFiles flexible enough to deal with any situation where you need to collect data. Like DataGrabber, SelectFiles has a *base_directory* parameter that allows you to specify a path that is common to all of the values in the *templates* dictionary. Additionally, as *glob* does not return a sorted list, there is also a *sort_filelist* option, taking a boolean, to control whether sorting should be applied (it is True by default).

The final input is *force_lists*, which controls how SelectFiles behaves in cases where only a single file matches the template. The default behavior is that when a template matches multiple files they are returned as a list, while a single file is returned as a string. There may be situations where you want to force the outputs to always be returned as a list (for example, you are writing a workflow that expects to operate on several runs of data, but some of your subjects only have a single run). In this case, *force_lists* can be used to tune the outputs of the interface. You can either use a boolean value, which will be applied to every output the interface has, or you can provide a list of the output fields that should be coerced to a list. Returning to our basic example, you may want to ensure that the *epi* files are returned as a list, but you only ever will have a single *T1* file. In this case, you would do

```
sf = SelectFiles(templates, force_lists=["epi"])
```

The Function Interface

Most Nipype interfaces provide access to external programs, such as FSL binaries or SPM routines. However, a special interface, `nipyne.interfaces.utility.Function`, allows you to wrap arbitrary Python code in the Interface framework and seamlessly integrate it into your workflows.

A Simple Function Interface

The most important component of a working Function interface is a Python function. There are several ways to associate a function with a Function interface, but the most common way will involve functions you code yourself as part of your Nipype scripts. Consider the following function:

```
def add_two(val):
    return val + 2
```

This simple function takes a value, adds 2 to it, and returns that new value.

Just as Nipype interfaces have inputs and outputs, Python functions have inputs, in the form of parameters or arguments, and outputs, in the form of their return values. When you define a Function interface object with an existing function, as in the case of `add_two()` above, you must pass the constructor information about the function's inputs, its outputs, and the function itself. For example,

```
from nipyne.interfaces.utility import Function
add_two_interface = Function(input_names=["val"],
                             output_names=["out_val"],
                             function=add_two)
```

Then you can set the inputs and run just as you would with any other interface:

```
add_two_interface.inputs.val = 2
res = add_two_interface.run()
print res.outputs.out_val
```

Which would print 4.

Note that, if you are working interactively, the Function interface is unable to use functions that are defined within your interpreter session. (Specifically, it can't use functions that live in the `__main__` namespace).

Using External Packages

Chances are, you will want to write functions that do more complicated processing, particularly using the growing stack of Python packages geared towards neuroimaging, such as [Nibabel](#), [Nipy](#), or [PyMVPA](#).

While this is completely possible (and, indeed, an intended use of the Function interface), it does come with one important constraint. The function code you write is executed in a standalone environment, which means that any external functions or classes you use have to be imported within the function itself:

```
def get_n_trs(in_file):
    import nibabel
    f = nibabel.load(in_file)
    return f.shape[-1]
```

Without explicitly importing Nibabel in the body of the function, this would fail.

Alternatively, it is possible to provide a list of strings corresponding to the imports needed to execute a function as a parameter of the *Function* constructor. This allows for the use of external functions that do not import all external definitions inside the function body.

Hello World - Function interface in a workflow

Contributed by: Hänel Nikolaus Valentin

The following snippet of code demonstrates the use of the function interface in the context of a workflow. Note the use of `import os` within the function as well as returning the absolute path from the Hello function. The *import* inside is necessary because functions are coded as strings and do not have to be on the PYTHONPATH. However any function called by this function has to be available on the PYTHONPATH. The *absolute path* is necessary because all workflow nodes are executed in their own directory and therefore there is no way of determining that the input file came from a different directory:

```
import nipy.pipeline.engine as pe
from nipy.interfaces.utility import Function

def Hello():
    import os
    from nipy import logging
    iflogger = logging.getLogger('interface')
    message = "Hello "
    file_name = 'hello.txt'
    iflogger.info(message)
    with open(file_name, 'w') as fp:
        fp.write(message)
    return os.path.abspath(file_name)

def World(in_file):
    from nipy import logging
    iflogger = logging.getLogger('interface')
    message = "World!"
    iflogger.info(message)
    with open(in_file, 'a') as fp:
        fp.write(message)
```

```

hello = pe.Node(name='hello',
                interface=Function(input_names=[],
                                   output_names=['out_file'],
                                   function=Hello))
world = pe.Node(name='world',
                interface=Function(input_names=['in_file'],
                                   output_names=[],
                                   function=World))

pipeline = pe.Workflow(name='nipytype_demo')
pipeline.connect([(hello, world, [('out_file', 'in_file')])])
pipeline.run()
pipeline.write_graph(graph2use='flat')

```

Advanced Use

To use an existing function object (as we have been doing so far) with a Function interface, it must be passed to the constructor. However, it is also possible to dynamically set how a Function interface will process its inputs using the special `function_str` input.

This input takes not a function object, but actually a single string that can be parsed to define a function. In the equivalent case to our example above, the string would be

```
add_two_str = "def add_two(val):\n    return val + 2\n"
```

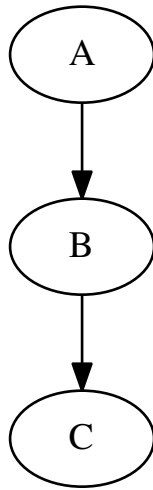
Unlike when using a function object, this input can be set like any other, meaning that you could write a function that outputs different function strings depending on some run-time contingencies, and connect that output the `function_str` input of a downstream Function interface.

MapNode, iterfield, and iterables explained

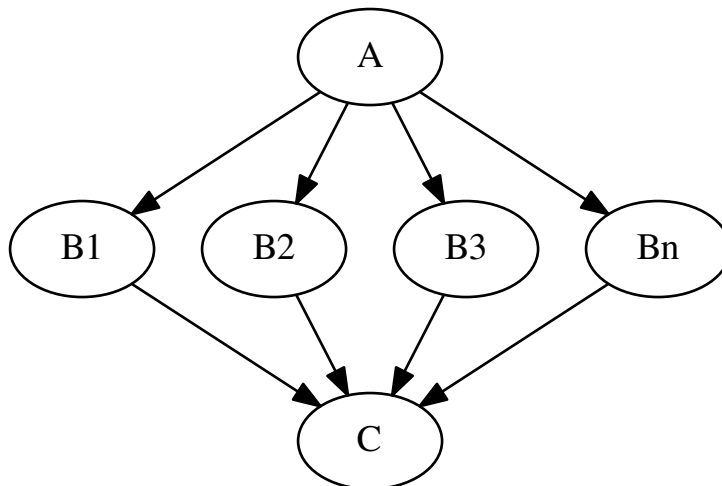
In this chapter we will try to explain the concepts behind MapNode, iterfield, and iterables.

MapNode and iterfield

Imagine that you have a list of items (lets say files) and you want to execute the same node on them (for example some smoothing or masking). Some nodes accept multiple files and do exactly the same thing on them, but some don't (they expect only one file). MapNode can solve this problem. Imagine you have the following workflow:



Node “A” outputs a list of files, but node “B” accepts only one file. Additionally “C” expects a list of files. What you would like is to run “B” for every file in the output of “A” and collect the results as a list and feed it to “C”. Something like this:



The code to achieve this is quite simple

```
import nipyype.pipeline.engine as pe
a = pe.Node(interface=A(), name="a")
b = pe.MapNode(interface=B(), name="b", iterfield=['in_file'])
c = pe.Node(interface=C(), name="c")
```

```
my_workflow = pe.Workflow(name="my_workflow")
my_workflow.connect([(a,b, [('out_files', 'in_file')]),
                      (b,c, [('out_file', 'in_files')])
                      ])
```

assuming that interfaces “A” and “C” have one input “in_files” and one output “out_files” (both lists of files). Interface “B” has single file input “in_file” and single file output “out_file”.

You probably noticed that you connect nodes as if “B” could accept and output list of files. This is because it is wrapped using MapNode instead of Node. This special version of node will (under the bonnet) create an instance of “B” for every item in the list from the input. The compulsory argument “iterfield” defines which input should it iterate over (for example in single file smooth interface you would like to iterate over input files not the smoothing width). At the end outputs are collected into a list again. In other words this is map and reduce scenario.

You might have also noticed that the iterfield arguments expects a list of input names instead of just one name. This suggests that there can be more than one! Even though a bit confusing this is true. You can specify more than one input to iterate over but the lists that you provide (for all the inputs specified in iterfield) have to have the same length. MapNode will then pair the parameters up and run the first instance with first set of parameters and second with second set of parameters. For example, this code:

```
b = pe.MapNode(interface=B(), name="b", iterfield=['in_file', 'n'])
b.inputs.in_file = ['file', 'another_file', 'different_file']
b.inputs.n = [1,2,3]
b.run()
```

is almost the same as running

```
b1 = pe.Node(interface=B(), name="b1")
b1.inputs.in_file = 'file'
b1.inputs.n = 1

b2 = pe.Node(interface=B(), name="b2")
b2.inputs.in_file = 'another_file'
b2.inputs.n = 2

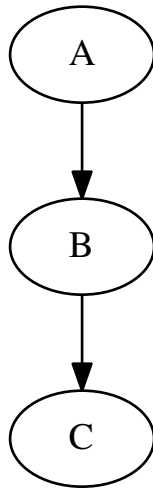
b3 = pe.Node(interface=B(), name="b3")
b3.inputs.in_file = 'different_file'
b3.inputs.n = 3
```

It is a rarely used feature, but you can sometimes find it useful.

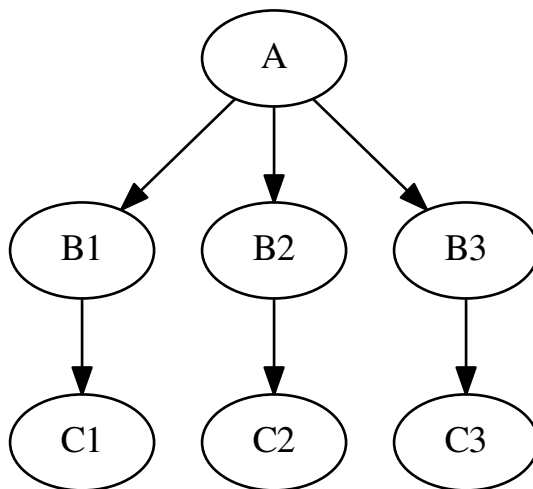
In more advanced applications it is useful to be able to iterate over items of nested lists (for example [[1,2],[3,4]]). MapNode allows you to do this with the “nested=True” parameter. Outputs will preserve the same nested structure as the inputs.

Iterables

Now imagine a different scenario. You have your workflow as before



and there are three possible values of one of the inputs node “B” you would like to investigate (for example width of 2,4, and 6 pixels of a smoothing node). You would like to see how different parameters in node “B” would influence everything that depends on its outputs (node “C” in our example). Therefore the new graph should look like this:



Of course you can do it manually by creating copies of all the nodes for different parameter set, but this can be very time consuming, especially when there are more than one node taking inputs from “B”. Luckily nipyype supports this scenario! Its called iterables and and you use it this way:

```
import nipyype.pipeline.engine as pe
a = pe.Node(interface=A(), name="a")
```



```
b = pe.Node(interface=B(), name="b")
b.iterables = ("n", [1, 2, 3])
c = pe.Node(interface=C(), name="c")

my_workflow = pe.Workflow(name="my_workflow")
my_workflow.connect([(a,b, [('out_file', 'in_file')]),
                     (b,c, [('out_file', 'in_file')])
                     ])
```

Assuming that you want to try out values 1, 2, and 3 of input “n” of the node “B”. This will also create three different versions of node “C” - each with inputs from instances of node “C” with different values of “n”.

Additionally, you can set multiple iterables for a node with a list of tuples in the above format.

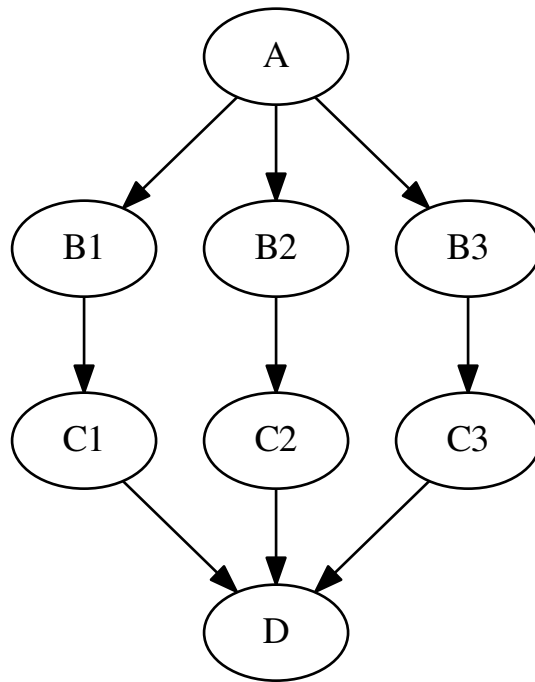
Iterables are commonly used to execute the same workflow for many subjects. Usually one parametrises DataGrabber node with subject ID. This is achieved by connecting an IdentityInterface in front of DataGrabber. When you set iterables of the IdentityInterface to the list of subjects IDs, the same workflow will be executed for every subject. See *fMRI: SPM, FSL* to see this pattern in action.

JoinNode, synchronize and itersource

The previous *MapNode, iterfield, and iterables explained* chapter described how to fork and join nodes using MapNode and iterables. In this chapter, we introduce features which build on these concepts to add workflow flexibility.

JoinNode, joinsource and joinfield

A `nipyte.pipeline.engine.JoinNode` generalizes MapNode to operate in conjunction with an upstream iterable node to reassemble downstream results, e.g.:



The code to achieve this is as follows:

```
import nipyype.pipeline.engine as pe
a = pe.Node(interface=A(), name="a")
b = pe.Node(interface=B(), name="b")
b.iterables = ("in_file", images)
c = pe.Node(interface=C(), name="c")
d = pe.JoinNode(interface=D(), joinsource="b",
                joinfield="in_files", name="d")

my_workflow = pe.Workflow(name="my_workflow")
my_workflow.connect([(a,b,['subject','subject'])],
                    (b,c,['out_file','in_file']),
                    (c,d,['out_file','in_files']))
])
```

This example assumes that interface “A” has one output *subject*, interface “B” has two inputs *subject* and *in_file* and one output *out_file*, interface “C” has one input *in_file* and one output *out_file*, and interface D has one list input *in_files*. The *images* variable is a list of three input image file names.

As with *iterables* and the MapNode *iterfield*, the *joinfield* can be a list of fields. Thus, the declaration in the previous example is equivalent to the following:

```
d = pe.JoinNode(interface=D(), joinsource="b",
                joinfield=["in_files"], name="d")
```

The *joinfield* defaults to all of the JoinNode input fields, so the declaration is also equivalent to the following:

```
d = pe.JoinNode(interface=D(), joinsource="b", name="d")
```

In this example, the node “c” *out_file* outputs are collected into the JoinNode “d” *in_files* input list. The *in_files*

order is the same as the upstream “b” node iterables order.

The JoinNode input can be filtered for unique values by specifying the *unique* flag, e.g.:

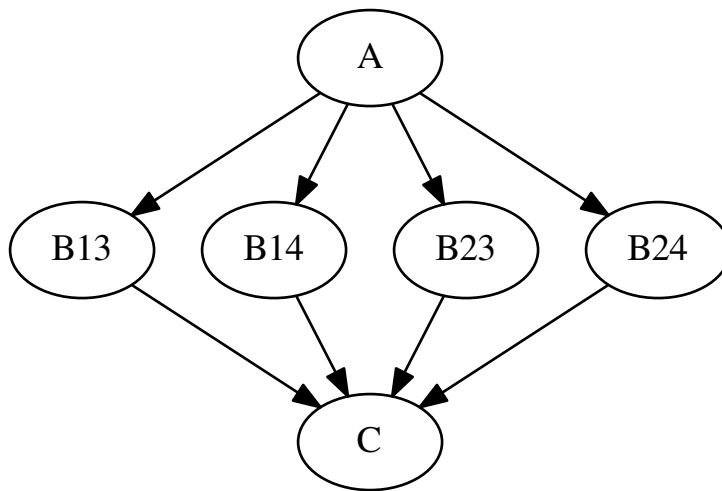
```
d = pe.JoinNode(interface=D(), joinsource="b", unique=True, name="d")
```

synchronize

The `nipyre.pipeline.engine.Node` *iterables* parameter can be a single field or a list of fields. If it is a list, then execution is performed over all permutations of the list items. For example:

```
b.iterables = [("m", [1, 2]), ("n", [3, 4])]
```

results in the execution graph:

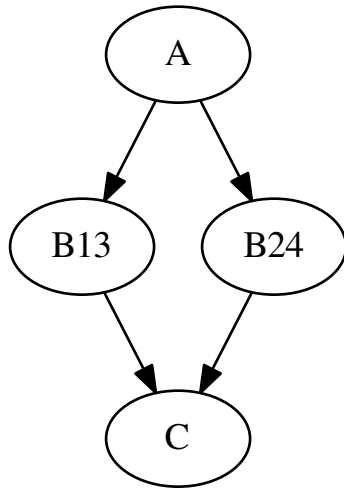


where “B13” has inputs $m = 1$, $n = 3$, “B14” has inputs $m = 1$, $n = 4$, etc.

The *synchronize* parameter synchronizes the iterables lists, e.g.:

```
b.iterables = [("m", [1, 2]), ("n", [3, 4])]
b.synchronize = True
```

results in the execution graph:



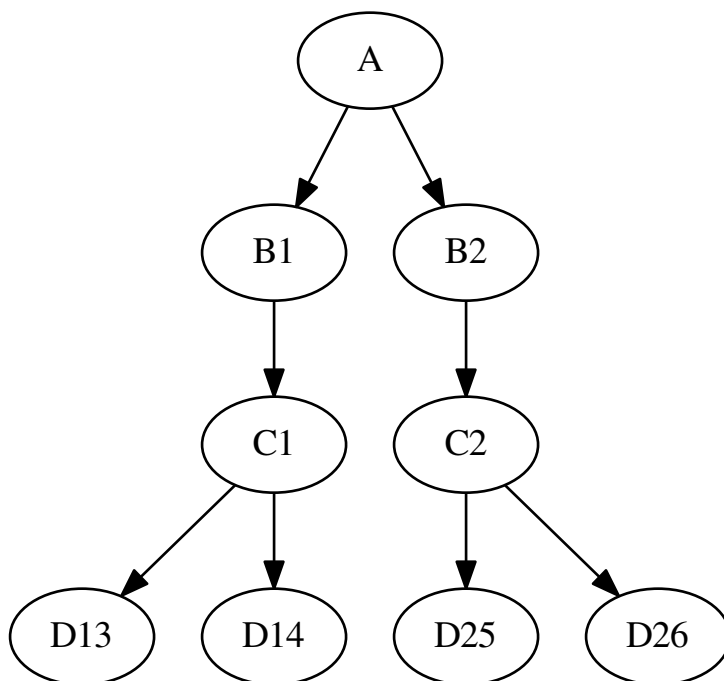
where the iterable inputs are selected in lock-step by index, i.e.:
(m, n) = (1, 3) and (2, 4)
for “B13” and “B24”, resp.

itersource

The *itersource* feature allows you to expand a downstream iterable based on a mapping of an upstream iterable. For example:

```
a = pe.Node(interface=A(), name="a")
b = pe.Node(interface=B(), name="b")
b.iterables = ("m", [1, 2])
c = pe.Node(interface=C(), name="c")
d = pe.Node(interface=D(), name="d")
d.itersource = ("b", "m")
d.iterables = [("n", {1:[3,4], 2:[5,6]})]
my_workflow = pe.Workflow(name="my_workflow")
my_workflow.connect([(a,b, [('out_file', 'in_file')]),
                    (b,c, [('out_file', 'in_file')]),
                    (c,d, [('out_file', 'in_file')])
                    ])
```

results in the execution graph:



In this example, all interfaces have input *in_file* and output *out_file*. In addition, interface “B” has input *m* and interface “D” has input *n*. A Python dictionary associates the “b” node input value with the downstream “d” node *n* iterable values.

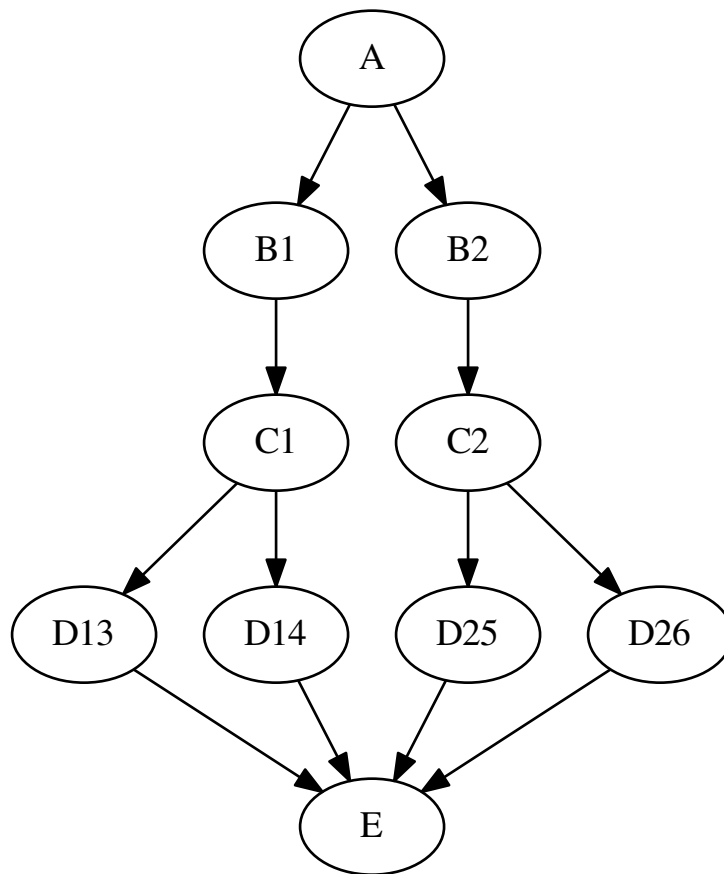
This example can be extended with a summary JoinNode:

```

e = pe.JoinNode(interface=E(), joinsource="d",
                joinfield="in_files", name="e")
my_workflow.connect(d, 'out_file',
                  e, 'in_files')

```

resulting in the graph:



The combination of iterables, MapNode, JoinNode, synchronize and itersource enables the creation of arbitrarily complex workflow graphs. The astute workflow builder will recognize that this flexibility is both a blessing and a curse. These advanced features are handy additions to the Nipype toolkit when used judiciously.

Model Specification for First Level fMRI Analysis

Nipype provides a general purpose model specification mechanism with specialized subclasses for package specific extensions.

General purpose model specification

The `SpecifyModel` provides a generic mechanism for model specification. A mandatory input called `subject_info` provides paradigm specification for each run corresponding to a subject. This has to be in the form of a `Bunch` or a list of `Bunch` objects (one for each run). Each `Bunch` object contains the following attributes.

Required for most designs

- `conditions` : list of names
- `onsets` : lists of onsets corresponding to each condition

- **durations** [lists of durations corresponding to each condition. Should be] left to a single 0 if all events are being modelled as impulses.

Optional

- **regressor_names** : list of names corresponding to each column. Should be None if automatically assigned.
- **regressors** : list of lists. values for each regressor - must correspond to the number of volumes in the functional run
- **amplitudes** [lists of amplitudes for each event. This will be ignored by] SPM's Level1Design.
The following two (tmod, pmod) will be ignored by any Level1Design class other than SPM:
- **tmod** [lists of conditions that should be temporally modulated. Should] default to None if not being used.
- **pmod** [list of Bunch corresponding to conditions]
 - name : name of parametric modulator
 - param : values of the modulator
 - poly : degree of modulation

An example Bunch definition:

```
from nipy.interfaces.base import Bunch
condnames = ['Tapping', 'Speaking', 'Yawning']
event_onsets = [[0, 10, 50], [20, 60, 80], [30, 40, 70]]
durations = [[0],[0],[0]]

subject_info = Bunch(conditions=condnames,
                     onsets = event_onsets,
                     durations = durations)
```

Alternatively, you can provide condition, onset, duration and amplitude information through event files. The event files have to be in 1,2 or 3 column format with the columns corresponding to Onsets, Durations and Amplitudes and they have to have the name event_name.run<anything else> e.g.: Words.run001.txt. The event_name part will be used to create the condition names. Words.run001.txt may look like:

```
# Word Onsets Durations
0    10
20   10
...
```

or with amplitudes:

```
# Word Onsets Durations Amplitudes
0    10    1
20   10    1
...
```

Together with this information, one needs to specify:

- whether the durations and event onsets are specified in terms of scan volumes or secs.
- the high-pass filter cutoff,
- the repetition time per scan
- functional data files corresponding to each run.

Optionally you can specify realignment parameters, outlier indices. Outlier files should contain a list of numbers, one per row indicating which scans should not be included in the analysis. The numbers are 0-based.

SPM specific attributes

in addition to the generic specification options, several SPM specific options can be provided. In particular, the subject_info function can provide temporal and parametric modulators in the Bunch attributes tmod and pmod. The following example adds a linear parametric modulator for speaking rate for the events specified earlier:

```

pmod = [None, Bunch(name=['Rate'], param=[[.300, .500, .600]],
                    poly=[1]), None]
subject_info = Bunch(conditions=condnames,
                    onsets = event_onsets,
                    durations = durations,
                    pmod = pmod)

```

SpecifySPMMModel also allows specifying additional components. If you have a study with multiple runs, you can choose to concatenate conditions from different runs. by setting the input option `concatenate_runs` to True. You can also choose to set the output options for this class to be in terms of ‘scans’.

Sparse model specification

In addition to standard models, SpecifySparseModel allows model generation for sparse and sparse-clustered acquisition experiments. Details of the model generation and utility are provided in [Ghosh et al. \(2009\)](#) [OHBM 2009](#).

Saving Workflows and Nodes to a file (experimental)

On top of the standard way of saving (i.e. serializing) objects in Python (see [pickle](#)) Nipype provides methods to turn Workflows and nodes into human readable code. This is useful if you want to save a Workflow that you have generated on the fly for future use.

To generate Python code for a Workflow use the export method:

```

from nipype.interfaces.fsl import BET, ImageMaths
from nipype.pipeline.engine import Workflow, Node, MapNode, format_node
from nipype.interfaces.utility import Function, IdentityInterface

bet = Node(BET(), name='bet')
bet.iterables = ('frac', [0.3, 0.4])

bet2 = MapNode(BET(), name='bet2', iterfield=['infile'])
bet2.iterables = ('frac', [0.4, 0.5])

maths = Node(ImageMaths(), name='maths')

def testfunc(in1):
    """dummy func
    """
    out = in1 + 'foo' + "out1"
    return out

funcnode = Node(Function(input_names=['a'], output_names=['output'],
    ↪function=testfunc),
                name='testfunc')
funcnode.inputs.in1 = '-sub'
func = lambda x: x

inode = Node(IdentityInterface(fields=['a']), name='inode')

wf = Workflow('testsave')
wf.add_nodes([bet2])
wf.connect(bet, 'mask_file', maths, 'in_file')
wf.connect(bet2, ('mask_file', func), maths, 'in_file2')
wf.connect(inode, 'a', funcnode, 'in1')
wf.connect(funcnode, 'output', maths, 'op_string')

```



```
wf.export()
```

This will create a file “outputtestsave.py” with the following content:

```
from nipy.pipeline.engine import Workflow, Node, MapNode
from nipy.interfaces.utility import IdentityInterface
from nipy.interfaces.utility import Function
from nipy.utils.misc import getsource
from nipy.interfaces.fsl.preprocess import BET
from nipy.interfaces.fsl.utils import ImageMaths
# Functions
func = lambda x: x
# Workflow
testsave = Workflow("testsave")
# Node: testsave.inode
inode = Node(IdentityInterface(fields=['a'], mandatory_inputs=True), name="inode")
# Node: testsave.testfunc
testfunc = Node(Function(input_names=['a'], output_names=['output']), name=
↳ "testfunc")
def testfunc_1(in1):
    """dummy func
    """
    out = in1 + 'foo' + "out1"
    return out

testfunc.inputs.function_str = getsource(testfunc_1)
testfunc.inputs.ignore_exception = False
testfunc.inputs.in1 = '-sub'
testsave.connect(inode, "a", testfunc, "in1")
# Node: testsave.bet2
bet2 = MapNode(BET(), iterfield=['infile'], name="bet2")
bet2.iterables = ('frac', [0.4, 0.5])
bet2.inputs.environ = {'FSLOUTPUTTYPE': 'NIFTI_GZ'}
bet2.inputs.ignore_exception = False
bet2.inputs.output_type = 'NIFTI_GZ'
bet2.inputs.terminal_output = 'stream'
# Node: testsave.bet
bet = Node(BET(), name="bet")
bet.iterables = ('frac', [0.3, 0.4])
bet.inputs.environ = {'FSLOUTPUTTYPE': 'NIFTI_GZ'}
bet.inputs.ignore_exception = False
bet.inputs.output_type = 'NIFTI_GZ'
bet.inputs.terminal_output = 'stream'
# Node: testsave.maths
maths = Node(ImageMaths(), name="maths")
maths.inputs.environ = {'FSLOUTPUTTYPE': 'NIFTI_GZ'}
maths.inputs.ignore_exception = False
maths.inputs.output_type = 'NIFTI_GZ'
maths.inputs.terminal_output = 'stream'
testsave.connect(bet2, ('mask_file', func), maths, "in_file2")
testsave.connect(bet, "mask_file", maths, "in_file")
testsave.connect(testfunc, "output", maths, "op_string")
```

The file is ready to use and includes all the necessary imports.

Using SPM with MATLAB Common Runtime

In order to use the standalone MCR version of `spm`, you need to ensure that the following commands are executed at the beginning of your script:

```
from nipyte.interfaces import spm
matlab_cmd = '/path/to/run_spm8.sh /path/to/Compiler_Runtime/v713/ script'
spm.SPMCommand.set_mlab_paths(matlab_cmd=matlab_cmd, use_mcr=True)
```

you can test by calling:

```
spm.SPMCommand().version
```

If you want to enforce the standalone MCR version of `spm` for `nipyte` globally, you can do so by setting the following environment variables:

SPMMCRCMD Specifies the command to use to run the `spm` standalone MCR version. You may still override the command as described above.

FORCE_SPMMCRC Set this to any value in order to enforce the use of `spm` standalone MCR version in `nipyte` globally. Technically, this sets the `use_mcr` flag of the `spm` interface to `True`.

Information about the MCR version of SPM8 can be found at:

<http://en.wikibooks.org/wiki/SPM/Standalone>

Using MIPAV, JIST, and CBS Tools

If you are trying to use MIPAV, JIST or CBS Tools interfaces you need to configure `CLASSPATH` environmental variable correctly. It needs to include extensions shipped with MIPAV, MIPAV itself and MIPAV plugins. For example:

In order to use the standalone MCR version of `spm`, you need to ensure that the following commands are executed at the beginning of your script:

```
# location of additional JAVA libraries to use
JAVALIB=/Applications/mipav/jre/Contents/Home/lib/ext/

# location of the MIPAV installation to use
MIPAV=/Applications/mipav
# location of the plugin installation to use
# please replace 'ThisUser' by your user name
PLUGINS=/Users/ThisUser/mipav/plugins

export CLASSPATH=$JAVALIB/*:$MIPAV:$MIPAV/lib/*:$PLUGINS
```

Running Nipyte Interfaces from the command line (nipyte_cmd)

The primary use of `Nipyte` is to build automated non-interactive pipelines. However, sometimes there is a need to run some interfaces quickly from the command line. This is especially useful when running Interfaces wrapping code that does not have command line equivalents (`nipy` or `SPM`). Being able to run `Nipyte` interfaces opens new possibilities such as inclusion of `SPM` processing steps in bash scripts.

To run `Nipyte` Interfaces you need to use the `nipyte_cmd` tool that should already be installed. The tool allows you to list Interfaces available in a certain package:

```
$nipyte_cmd nipyte.interfaces.nipy

Available Interfaces:
  SpaceTimeRealigner
  Similarity
```

```

ComputeMask
FitGLM
EstimateContrast
FmriRealign4d

```

After selecting a particular Interface you can learn what inputs it requires:

```

$nipype_cmd nipype.interfaces.nipy ComputeMask --help

usage:nipype_cmd nipype.interfaces.nipy ComputeMask [-h] [--M M] [--cc CC]
                                                    [--ignore_exception IGNORE_
↳EXCEPTION]
                                                    [--m M]
                                                    [--reference_volume_
↳REFERENCE_VOLUME]
                                                    mean_volume

Run ComputeMask

positional arguments:
  mean_volume          mean EPI image, used to compute the threshold for the
                        mask

optional arguments:
  -h, --help          show this help message and exit
  --M M              upper fraction of the histogram to be discarded
  --cc CC            Keep only the largest connected component
  --ignore_exception IGNORE_EXCEPTION
                        Print an error message instead of throwing an
                        exception in case the interface fails to run
  --m M              lower fraction of the histogram to be discarded
  --reference_volume REFERENCE_VOLUME
                        reference volume used to compute the mask. If none is
                        give, the mean volume is used.

```

Finally you can run the Interface:

```
$nipype_cmd nipype.interfaces.nipy ComputeMask mean.nii.gz
```

All that from the command line without having to start python interpreter manually.

Using Nipype with Amazon Web Services (AWS)

Several groups have been successfully using Nipype on AWS. This procedure involves setting a temporary cluster using StarCluster and potentially transferring files to/from S3. The latter is supported by Nipype through DataSink and S3DataGrabber.

Using DataSink with S3

The DataSink class now supports sending output data directly to an AWS S3 bucket. It does this through the introduction of several input attributes to the DataSink interface and by parsing the *base_directory* attribute. This class uses the `boto3` and `botocore` Python packages to interact with AWS. To configure the DataSink to write data to S3, the user must set the *base_directory* property to an S3-style filepath. For example:

```

import nipype.interfaces.io as nio
ds = nio.DataSink()
ds.inputs.base_directory = 's3://mybucket/path/to/output/dir'

```

With the “s3://” prefix in the path, the DataSink knows that the output directory to send files is on S3 in the bucket “mybucket”. “path/to/output/dir” is the relative directory path within the bucket “mybucket” where output data will be uploaded to (NOTE: if the relative path specified contains folders that don’t exist in the bucket, the DataSink will create them). The DataSink treats the S3 base directory exactly as it would a local directory, maintaining support for containers, substitutions, subfolders, “.” notation, etc to route output data appropriately. There are four new attributes introduced with S3-compatibility: `creds_path`, `encrypt_bucket_keys`, `local_copy`, and `bucket`.

```
ds.inputs.creds_path = '/home/user/aws_creds/credentials.csv'
ds.inputs.encrypt_bucket_keys = True
ds.local_copy = '/home/user/workflow_outputs/local_backup'
```

`creds_path` is a file path where the user’s AWS credentials file (typically a csv) is stored. This credentials file should contain the AWS access key id and secret access key and should be formatted as one of the following (these formats are how Amazon provides the credentials file by default when first downloaded).

Root-account user:

```
AWSAccessKeyId=ABCDEFGHIJKLMNPO
AWSecretKey=zyx123wvu456/ABC890+gHiJk
```

IAM-user:

```
User Name,Access Key Id,Secret Access Key
"username",ABCDEFGHIJKLMNPO,zyx123wvu456/ABC890+gHiJk
```

The `creds_path` is necessary when writing files to a bucket that has restricted access (almost no buckets are publicly writable). If `creds_path` is not specified, the DataSink will check the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` environment variables and use those values for bucket access.

`encrypt_bucket_keys` is a boolean flag that indicates whether to encrypt the output data on S3, using server-side AES-256 encryption. This is useful if the data being output is sensitive and one desires an extra layer of security on the data. By default, this is turned off.

`local_copy` is a string of the filepath where local copies of the output data are stored in addition to those sent to S3. This is useful if one wants to keep a backup version of the data stored on their local computer. By default, this is turned off.

`bucket` is a boto3 Bucket object that the user can use to overwrite the bucket specified in their `base_directory`. This can be useful if one has to manually create a bucket instance on their own using special credentials (or using a mock server like `fakes3`). This is typically used for developers unit-testing the DataSink class. Most users do not need to use this attribute for actual workflows. This is an optional argument. Finally, the user needs only to specify the input attributes for any incoming data to the node, and the outputs will be written to their S3 bucket.

```
workflow.connect(inputnode, 'subject_id', ds, 'container')
workflow.connect(realigner, 'realigned_files', ds, 'motion')
```

So, for example, outputs for sub001’s `realigned_file1.nii.gz` will be in: `s3://mybucket/path/to/output/dir/sub001/motion/realigned_file1.nii.gz`

Using S3DataGrabber

Coming soon...

Resource Scheduling and Profiling with Nipyte

The latest version of Nipyte supports system resource scheduling and profiling. These features allows users to ensure high throughput of their data processing while also controlling the amount of computing resources a given workflow will use.

Specifying Resources in the Node Interface

Each Node instance interface has two parameters that specify its expected thread and memory usage: `num_threads` and `estimated_memory_gb`. If a particular node is expected to use 8 threads and 2 GB of memory:

```
import nipyte.pipeline.engine as pe
node = pe.Node()
node.interface.num_threads = 8
node.interface.estimated_memory_gb = 2
```

If the resource parameters are never set, they default to being 1 thread and 1 GB of RAM.

Resource Scheduler

The `MultiProc` workflow plugin schedules node execution based on the resources used by the current running nodes and the total resources available to the workflow. The plugin utilizes the plugin arguments `n_procs` and `memory_gb` to set the maximum resources a workflow can utilize. To limit a workflow to using 8 cores and 10 GB of RAM:

```
args_dict = {'n_procs' : 8, 'memory_gb' : 10}
workflow.run(plugin='MultiProc', plugin_args=args_dict)
```

If these values are not specifically set then the plugin will assume it can use all of the processors and memory on the system. For example, if the machine has 16 cores and 12 GB of RAM, the workflow will internally assume those values for `n_procs` and `memory_gb`, respectively.

The plugin will then queue eligible nodes for execution based on their expected usage via the `num_threads` and `estimated_memory_gb` interface parameters. If the plugin sees that only 3 of its 8 processors and 4 GB of its 10 GB of RAM are being used by running nodes, it will attempt to execute the next available node as long as its `num_threads` ≤ 5 and `estimated_memory_gb` ≤ 6 . If this is not the case, it will continue to check every available node in the queue until it sees a node that meets these conditions, or it waits for an executing node to finish to earn back the necessary resources. The priority of the queue is highest for nodes with the most `estimated_memory_gb` followed by nodes with the most expected `num_threads`.

Runtime Profiler and using the Callback Log

It is not always easy to estimate the amount of resources a particular function or command uses. To help with this, Nipyte provides some feedback about the system resources used by every node during workflow execution via the built-in runtime profiler. The runtime profiler is automatically enabled if the `psutil` Python package is installed and found on the system.

If the package is not found, the workflow will run normally without the runtime profiler.

The runtime profiler records the number of threads and the amount of memory (GB) used as `runtime_threads` and `runtime_memory_gb` in the Node's `result.runtime` attribute. Since the node object is pickled and written to disk in its working directory, these values are available for analysis after node or workflow execution by manually parsing the pickle file contents.

Nipyte also provides a logging mechanism for saving node runtime statistics to a JSON-style log file via the `log_nodes_cb` logger function. This is enabled by setting the `status_callback` parameter to point to this function in the `plugin_args` when using the `MultiProc` plugin.

```
from nipyte.pipeline.plugins.callback_log import log_nodes_cb
args_dict = {'n_procs' : 8, 'memory_gb' : 10, 'status_callback' : log_nodes_cb}
```

To set the filepath for the callback log the `'callback'` logger must be configured.

```
# Set path to log file
import logging
callback_log_path = '/home/user/run_stats.log'
logger = logging.getLogger('callback')
```

```
logger.setLevel(logging.DEBUG)
handler = logging.FileHandler(callback_log_path)
logger.addHandler(handler)
```

Finally, the workflow can be run.

```
workflow.run(plugin='MultiProc', plugin_args=args_dict)
```

After the workflow finishes executing, the log file at “/home/user/run_stats.log” can be parsed for the runtime statistics. Here is an example of what the contents would look like:

```
{ "name": "resample_node", "id": "resample_node",
  "start": "2016-03-11 21:43:41.682258",
  "estimated_memory_gb": 2, "num_threads": 1 }
{ "name": "resample_node", "id": "resample_node",
  "finish": "2016-03-11 21:44:28.357519",
  "estimated_memory_gb": 2, "num_threads": 1,
  "runtime_threads": 3, "runtime_memory_gb": "1.118469238281" }
```

Here it can be seen that the number of threads was underestimated while the amount of memory needed was overestimated. The next time this workflow is run the user can change the node interface `num_threads` and `estimated_memory_gb` parameters to reflect this for a higher pipeline throughput. Note, sometimes the “runtime_threads” value is higher than expected, particularly for multi-threaded applications. Tools can implement multi-threading in different ways under-the-hood; the profiler merely traverses the process tree to return all running threads associated with that process, some of which may include active thread-monitoring daemons or transient processes.

Visualizing Pipeline Resources

Nipyne provides the ability to visualize the workflow execution based on the runtimes and system resources each node takes. It does this using the log file generated from the callback logger after workflow execution - as shown above. The `pandas` Python package is required to use this feature.

```
from nipyne.pipeline.plugins.callback_log import log_nodes_cb
args_dict = {'n_procs' : 8, 'memory_gb' : 10, 'status_callback' : log_nodes_cb}
workflow.run(plugin='MultiProc', plugin_args=args_dict)

# ...workflow finishes and writes callback log to '/home/user/run_stats.log'

from nipyne.utils.draw_gantt_chart import generate_gantt_chart
generate_gantt_chart('/home/user/run_stats.log', cores=8)
# ...creates gantt chart in '/home/user/run_stats.log.html'
```

The `generate_gantt_chart` function will create an html file that can be viewed in a browser. Below is an example of the gantt chart displayed in a web browser. Note that when the cursor is hovered over any particular node bubble or resource bubble, some additional information is shown in a pop-up.

-



Sphinx extensions

To help users document their *Nipype*-based code, the software is shipped with a set of extensions (currently only one) to customize the appearance and simplify the generation process.

`nipy.sphinxext.plot_workflow` – Workflow plotting extension

A directive for including a nipype workflow graph in a Sphinx document.

This code is forked from the `plot_figure` sphinx extension of matplotlib.

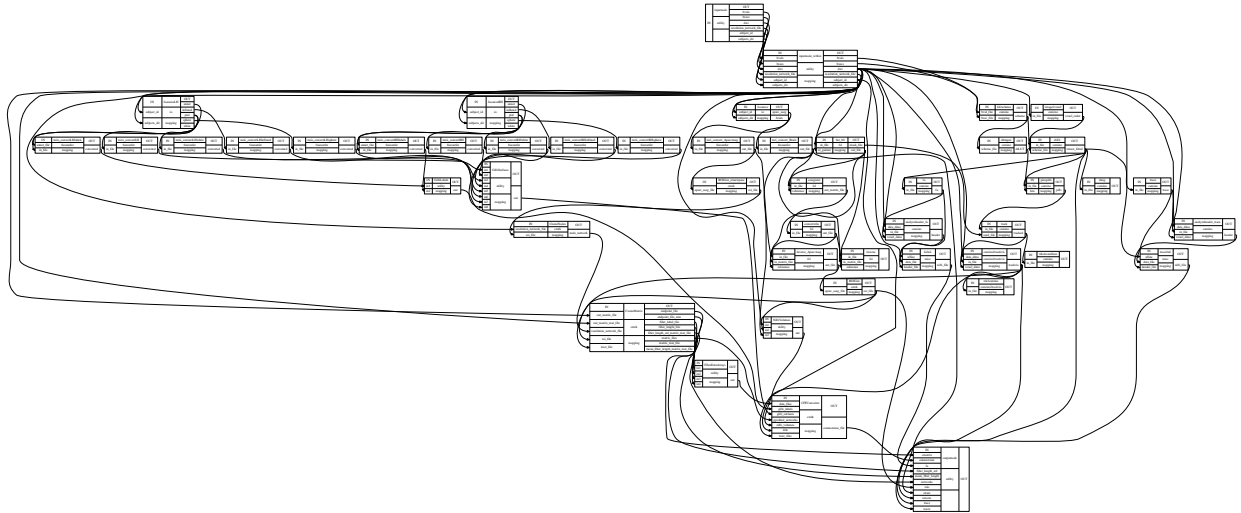
By default, in HTML output, *workflow* will include a .png file with a link to a high-res .png. In LaTeX output, it will include a .pdf. The source code for the workflow may be included as **inline content** to the directive

workflow:

```
.. workflow ::
    :graph2use: flat
    :simple_form: no

    from nipyne.workflows.dmri.camino.connectivity_mapping import create_
    ↪connectivity_pipeline
    wf = create_connectivity_pipeline()
```

For example, the following graph has been generated inserting the previous code block in this documentation:



Options

The **workflow** directive supports the following options:

- graph2use** [{‘hierarchical’, ‘colored’, ‘flat’, ‘orig’, ‘exec’}] Specify the type of graph to be generated.
- simple_form**: **bool** Whether the graph will be in detailed or simple form.
- format** [{‘python’, ‘doctest’}] Specify the format of the input
- include-source** [bool] Whether to display the source code. The default can be changed using the *workflow_include_source* variable in *conf.py*
- encoding** [str] If this source file is in a non-UTF8 or non-ASCII encoding, the encoding must be specified using the *:encoding:* option. The encoding will not be inferred using the `-- coding --` metacomment.

Additionally, this directive supports all of the options of the *image* directive, except for *target* (since workflow will add its own target). These include *alt*, *height*, *width*, *scale*, *align* and *class*.

Configuration options

The **workflow** directive has the following configuration options:

- graph2use** Select a graph type to use
- simple_form** determines if the node name shown in the visualization is either of the form `nodename (package)` when set to True or `nodename.Class.package` when set to False.
- wf_include_source** Default value for the include-source option
- wf_html_show_source_link** Whether to show a link to the source in HTML.
- wf_pre_code** Code that should be executed before each workflow.
- wf_basedir** Base directory, to which *workflow::* file names are relative to. (If None or empty, file names are relative to the directory where the file containing the directive is.)
- wf_formats**

File formats to generate. List of tuples or strings:: [(suffix, dpi), suffix, ...]

that determine the file format and the DPI. For entries whose DPI was omitted, sensible defaults are chosen. When passing from the command line through `sphinx_build` the list should be passed as `suffix:dpi,suffix:dpi,`

wf_html_show_formats Whether to show links to the files in HTML.

wf_rcparams A dictionary containing any non-standard rcParams that should be applied before each workflow.

wf_apply_rcparams By default, rcParams are applied when *context* option is not used in a workflow directive. This configuration option overrides this behavior and applies rcParams before each workflow.

wf_working_directory By default, the working directory will be changed to the directory of the example, so the code can get at its data files, if any. Also its path will be added to *sys.path* so it can import any helper modules sitting beside it. This configuration option can be used to specify a central directory (also added to *sys.path*) where data files and helper modules for all code are located.

wf_template Provide a customized template for preparing restructured text.

Changes in Nipype

0.13.1 (May 20, 2017)

- FIX: Make release compatible with conda-forge build process (<https://github.com/nipy/nipype/pull/2017>)
- ENH: Update some minimum versions in compliance with Debian Jessie (<https://github.com/nipy/nipype/pull/2017>)
- ENH: Circle builds use cached docker layers (<https://github.com/nipy/nipype/pull/2017>)
- ENH: Base docker to use FS6 and ANTS 2.2.0 (<https://github.com/nipy/nipype/pull/2024>)
- FIX: Mailmap and contributor acknowledgment (<https://github.com/nipy/nipype/pull/2017>)
- FIX: Preserve node properties in sub nodes of MapNode (<https://github.com/nipy/nipype/pull/2019>)
- FIX: Fix interfaces 3DUnifize, ICA_AROMA, BinaryMaths, RegAverage, BBRegister, AffineInitializer (<https://github.com/nipy/nipype/pull/2025>, <https://github.com/nipy/nipype/pull/2027>, <https://github.com/nipy/nipype/pull/2036>, <https://github.com/nipy/nipype/pull/2037>, <https://github.com/nipy/nipype/pull/2031>, <https://github.com/nipy/nipype/pull/2010>)
- ENH: Add Anisotropic Power interface (<https://github.com/nipy/nipype/pull/2039>)
- FIX: Bayesian estimation in SPM (<https://github.com/nipy/nipype/pull/2030>)

0.13.0 (May 11, 2017)

- ENH: Multi-stage recon-all directives (<https://github.com/nipy/nipype/pull/1991>)
- FIX: FEAT “folder does not exist” error (<https://github.com/nipy/nipype/pull/2000>)
- ENH: Niftyfit interfaces (<https://github.com/nipy/nipype/pull/1910>)
- FIX: Define ANTSPATH for BrainExtraction automatically (<https://github.com/nipy/nipype/pull/1986>)
- ENH: New trait for imaging files (<https://github.com/nipy/nipype/pull/1949>)
- ENH: Niftyseg interfaces (<https://github.com/nipy/nipype/pull/1911>)
- ENH: Niftyreg interfaces (<https://github.com/nipy/nipype/pull/1913>)
- MRG: Allow more support for CLI (<https://github.com/nipy/nipype/pull/1908>)
- ENH: 3dQwarpPlusMinus interface (<https://github.com/nipy/nipype/pull/1974>)
- FIX: PY3.6 support (<https://github.com/nipy/nipype/pull/1977>)
- FIX: PY3 and stream fixes for MRTrix2TrackVis (<https://github.com/nipy/nipype/pull/1804>)
- ENH: More mask options for CompCor interfaces (<https://github.com/nipy/nipype/pull/1968> + <https://github.com/nipy/nipype/pull/1992>)
- ENH: Additional TOPUP outputs (<https://github.com/nipy/nipype/pull/1976>)
- ENH: Additional Eddy flags (<https://github.com/nipy/nipype/pull/1967>)
- ENH: ReconAll handlers for less common cases (<https://github.com/nipy/nipype/pull/1966>)

- ENH: FreeSurferSource now finds graymid/midthickness surfs (<https://github.com/nipy/nipype/pull/1972>)
- ENH: Additional fsmaths dimensional reduction operations (<https://github.com/nipy/nipype/pull/1956>)
- ENH: More options for RobustFOV interface (<https://github.com/nipy/nipype/pull/1923>)
- ENH: Add MRIsCombine to FreeSurfer utils (<https://github.com/nipy/nipype/pull/1948>)
- FIX: Level1Design EV parameter substitution (<https://github.com/nipy/nipype/pull/1953>)
- FIX: Dcm2niix outputs can be uncompressed (<https://github.com/nipy/nipype/pull/1951>)
- FIX: Ensure build fails in Circle when tests fail (<https://github.com/nipy/nipype/pull/1981>)
- ENH: Add interface to antsAffineInitializer (<https://github.com/nipy/nipype/pull/1980>)
- ENH: AFNI motion parameter support for FrameWiseDisplacement (<https://github.com/nipy/nipype/pull/1840>)
- ENH: Add ANTs KellyKapowski interface (<https://github.com/nipy/nipype/pull/1845>)
- FIX: AFNI interface bug setting OMP_NUM_THREADS to 1 (<https://github.com/nipy/nipype/pull/1728>)
- FIX: Select Eddy run command at runtime (<https://github.com/nipy/nipype/pull/1871>)
- FIX: Increase FLIRT's flexibility with apply_xfm (<https://github.com/nipy/nipype/pull/1875>)
- DOC: Update FSL preprocess docstrings (<https://github.com/nipy/nipype/pull/1881>)
- ENH: Support GIFTI outputs in SampleToSurface (<https://github.com/nipy/nipype/pull/1886>)
- FIX: Configparser differences between PY2 and PY3 (<https://github.com/nipy/nipype/pull/1890>)
- ENH: Add mris_expand interface (<https://github.com/nipy/nipype/pull/1893>)
- FIX: Split over-eager globs in FreeSurferSource (<https://github.com/nipy/nipype/pull/1894>)
- FIX: Store undefined by default so that xor checks don't trip (<https://github.com/nipy/nipype/pull/1903>)
- FIX: Gantt chart generator PY3 compatibility (<https://github.com/nipy/nipype/pull/1907>)
- FIX: Add DOF and -fsl-dof options to BBRegister (<https://github.com/nipy/nipype/pull/1917>)
- ENH: Auto-derive input_names in Function (<https://github.com/nipy/nipype/pull/1918>)
- FIX: Minor fixes for NonSteadyStateDetector (<https://github.com/nipy/nipype/pull/1926>)
- DOC: Add duecredit references for AFNI and FSL (<https://github.com/nipy/nipype/pull/1930>)
- ENH: Added zenodo (<https://zenodo.org/>) file (<https://github.com/nipy/nipype/pull/1924>)
- ENH: Disable symlinks on CIFS filesystems (<https://github.com/nipy/nipype/pull/1941>)
- ENH: Sphinx extension to plot workflows (<https://github.com/nipy/nipype/pull/1896>)
- ENH: Added non-steady state detector for EPI data (<https://github.com/nipy/nipype/pull/1839>)
- ENH: Enable new BBRegister init options for FSv6+ (<https://github.com/nipy/nipype/pull/1811>)
- REF: Splits nipype.interfaces.utility into base, csv, and wrappers (<https://github.com/nipy/nipype/pull/1828>)
- FIX: Makespec now runs with nipype in current directory (<https://github.com/nipy/nipype/pull/1813>)
- FIX: Flexible nifti opening with mmap if Numpy < 1.12.0 (<https://github.com/nipy/nipype/pull/1796> + <https://github.com/nipy/nipype/pull/1831>)
- ENH: DVARS includes intensity normalization feature - turned on by default (<https://github.com/nipy/nipype/pull/1827>)
- FIX: DVARS is correctly using sum of squares instead of standard deviation (<https://github.com/nipy/nipype/pull/1827>)
- ENH: Refactoring of nipype.interfaces.utility (<https://github.com/nipy/nipype/pull/1828>)
- FIX: CircleCI were failing silently. Some fixes to tests (<https://github.com/nipy/nipype/pull/1833>)
- FIX: Issues in Docker image permissions, and docker documentation (<https://github.com/nipy/nipype/pull/1825>)
- **ENH: Revised all Dockerfiles and automated deployment to Docker Hub** from CircleCI (<https://github.com/nipy/nipype/pull/1815>)
- ENH: Update ReconAll interface for FreeSurfer v6.0.0 (<https://github.com/nipy/nipype/pull/1790>)
- FIX: Cast DVARS float outputs to avoid mmap error (<https://github.com/nipy/nipype/pull/1777>)
- FIX: FSL FNIRT intensity mapping files (<https://github.com/nipy/nipype/pull/1799>)
- ENH: Additional outputs generated by FSL EDDY (<https://github.com/nipy/nipype/pull/1793>)
- TST: Parallelize CircleCI build across 4 containers (<https://github.com/nipy/nipype/pull/1769>)

0.13.0-rc1 (January 4, 2017)

- FIX: Compatibility with traits 4.6 (<https://github.com/nipy/nipype/pull/1770>)
- FIX: Multiproc deadlock (<https://github.com/nipy/nipype/pull/1756>)
- TST: Replace nose and unittest with pytest (<https://github.com/nipy/nipype/pull/1722>, <https://github.com/nipy/>)

[nipyre/pull/1751](#))

- FIX: Semaphore capture using MultiProc plugin (<https://github.com/nipy/nipyre/pull/1689>)
- REF: Refactor AFNI interfaces (<https://github.com/nipy/nipyre/pull/1678>, <https://github.com/nipy/nipyre/pull/1680>)
- ENH: Move nipyre commands to group command using click (<https://github.com/nipy/nipyre/pull/1608>)
- FIX: AFNI Retroicor interface fixes (<https://github.com/nipy/nipyre/pull/1669>)
- FIX: Minor errors after migration to setuptools (<https://github.com/nipy/nipyre/pull/1671>)
- ENH: Add AFNI 3dNote interface (<https://github.com/nipy/nipyre/pull/1637>)
- ENH: Abandon distutils, only use setuptools (<https://github.com/nipy/nipyre/pull/1627>)
- FIX: Minor bugfixes related to unicode literals (<https://github.com/nipy/nipyre/pull/1656>)
- TST: Automatic retries in travis (<https://github.com/nipy/nipyre/pull/1659/files>)
- ENH: Add signal extraction interface (<https://github.com/nipy/nipyre/pull/1647>)
- ENH: Add a DVARS calculation interface (<https://github.com/nipy/nipyre/pull/1606>)
- ENH: New interface to b0calc of FSL-POSSUM (<https://github.com/nipy/nipyre/pull/1399>)
- ENH: Add CompCor (<https://github.com/nipy/nipyre/pull/1599>)
- ENH: Add duecredit entries (<https://github.com/nipy/nipyre/pull/1466>)
- FIX: Python 3 compatibility fixes (<https://github.com/nipy/nipyre/pull/1572>)
- REF: Improved PEP8 compliance for fsl interfaces (<https://github.com/nipy/nipyre/pull/1597>)
- REF: Improved PEP8 compliance for spm interfaces (<https://github.com/nipy/nipyre/pull/1593>)
- TST: Replaced coveralls with codecov (<https://github.com/nipy/nipyre/pull/1609>)
- ENH: More BrainSuite interfaces (<https://github.com/nipy/nipyre/pull/1554>)
- ENH: Convenient load/save of interface inputs (<https://github.com/nipy/nipyre/pull/1591>)
- ENH: Add a Framewise Displacement calculation interface (<https://github.com/nipy/nipyre/pull/1604>)
- FIX: Use builtins open and unicode literals for py3 compatibility (<https://github.com/nipy/nipyre/pull/1572>)
- TST: reduce the size of docker images & use tags for images (<https://github.com/nipy/nipyre/pull/1564>)
- ENH: Implement missing inputs/outputs in FSL AvScale (<https://github.com/nipy/nipyre/pull/1563>)
- FIX: Fix symlink test in copyfile (<https://github.com/nipy/nipyre/pull/1570>, <https://github.com/nipy/nipyre/pull/1586>)
- ENH: Added support for custom job submission check in SLURM (<https://github.com/nipy/nipyre/pull/1582>)
- **ENH: Added ANTs interface CreateJacobianDeterminantImage; replaces deprecated JacobianDeterminant** (<https://github.com/nipy/nipyre/pull/1654>)

Release 0.12.1 (August 3, 2016)

- FIX: runtime profiling is optional and off by default (<https://github.com/nipy/nipyre/pull/1561>)
- TST: circle CI tests run with docker (<https://github.com/nipy/nipyre/pull/1541>)
- FIX: workflow export functions without import error (<https://github.com/nipy/nipyre/pull/1552>)

Release 0.12.0 (July 12, 2016)

- ENH: New interface for Bruker to Nifti converter (<https://github.com/nipy/nipyre/pull/1523>)
- FIX: output file naming for FIRST outputs (<https://github.com/nipy/nipyre/pull/1524>)
- ENH: Adds *fslmaths -Tstd* to maths interfaces (<https://github.com/nipy/nipyre/pull/1518>)
- FIX: Selecting “gamma” in FSL Level1Design now does what the name says (<https://github.com/nipy/nipyre/pull/1500>)
- ENH: Added grad_dev input to fsl.dti.bedpostx5 interface (<https://github.com/nipy/nipyre/pull/1493>)
- ENH: ResourceMultiProc plugin to support resource allocation (<https://github.com/nipy/nipyre/pull/1372>)
- ENH: Added dcm2niix interface (<https://github.com/nipy/nipyre/pull/1435>)
- ENH: Add nipyre_crash_search command (<https://github.com/nipy/nipyre/pull/1422>)
- ENH: Created interface for BrainSuite Cortical Surface Extraction command line tools (<https://github.com/nipy/nipyre/pull/1305>)
- FIX: job execution on systems/approaches where locale is undefined (<https://github.com/nipy/nipyre/pull/1401>)
- FIX: Clean up byte/unicode issues using subprocess (<https://github.com/nipy/nipyre/pull/1394>)

- FIX: Prevent crash when tvtk is loaded - ETS_TOOLKIT=null (<https://github.com/nipy/nipype/pull/973>)
- **ENH: New interfaces in dipy: RESTORE, EstimateResponseSH, CSD and StreamlineTractography** (<https://github.com/nipy/nipype/pull/1090>)
- **ENH: Added interfaces of AFNI** (<https://github.com/nipy/nipype/pull/1360>, <https://github.com/nipy/nipype/pull/1361>, <https://github.com/nipy/nipype/pull/1382>)
- ENH: Provides a Nipype wrapper for antsJointFusion (<https://github.com/nipy/nipype/pull/1351>)
- ENH: Added support for PETPVC (<https://github.com/nipy/nipype/pull/1335>)
- ENH: Merge S3DataSink into DataSink, added AWS documentation (<https://github.com/nipy/nipype/pull/1316>)
- TST: Cache APT in CircleCI (<https://github.com/nipy/nipype/pull/1333>)
- ENH: Add new flags to the BRAINSABC for new features (<https://github.com/nipy/nipype/pull/1322>)
- ENH: Provides a Nipype wrapper for ANTs DenoiseImage (<https://github.com/nipy/nipype/pull/1291>)
- FIX: Minor bugfix logging hash differences (<https://github.com/nipy/nipype/pull/1298>)
- FIX: Use released Prov python library (<https://github.com/nipy/nipype/pull/1279>)
- ENH: Support for Python 3 (<https://github.com/nipy/nipype/pull/1221>)
- FIX: VTK version check missing when using tvtk (<https://github.com/nipy/nipype/pull/1219>)
- ENH: Added an OAR scheduler plugin (<https://github.com/nipy/nipype/pull/1259>)
- ENH: New ANTs interface: antsBrainExtraction (<https://github.com/nipy/nipype/pull/1231>)
- API: Default model level for the bedpostx workflow has been set to “2” following FSL 5.0.9 lead
- ENH: New interfaces for interacting with AWS S3: S3DataSink and S3DataGrabber (<https://github.com/nipy/nipype/pull/1201>)
- ENH: Interfaces for MINC tools (<https://github.com/nipy/nipype/pull/1304>)
- FIX: Use realpath to determine hard link source (<https://github.com/nipy/nipype/pull/1388>)
- FIX: Correct linking/copying fallback behavior (<https://github.com/nipy/nipype/pull/1391>)
- ENH: Nipype workflow and interfaces for FreeSurfer’s recon-all (<https://github.com/nipy/nipype/pull/1326>)
- FIX: Permit relative path for concatenated_file input to Concatenate() (<https://github.com/nipy/nipype/pull/1411>)
- ENH: Makes ReconAll workflow backwards compatible with FreeSurfer 5.3.0 (<https://github.com/nipy/nipype/pull/1434>)
- **ENH: Added interfaces for AFNI 3dDegreeCentrality, 3dECM, 3dLFCD, 3dClipLevel, 3dmask_tool, and 3dSeg** (<https://github.com/nipy/nipype/pull/1460>)

Release 0.11.0 (September 15, 2015)

- API: Change how hash values are computed (<https://github.com/nipy/nipype/pull/1174>)
- **ENH: New algorithm: mesh.WarpPoints applies displacements fields to point sets** (<https://github.com/nipy/nipype/pull/889>).
- ENH: New interfaces for MRTrix3 (<https://github.com/nipy/nipype/pull/1126>)
- ENH: New option in afni.3dRefit - zdel, ydel, zdel etc. (<https://github.com/nipy/nipype/pull/1079>)
- FIX: ants.Registration composite transform outputs are no longer returned as lists (<https://github.com/nipy/nipype/pull/1183>)
- **BUG: ANTs Registration interface failed with multi-modal inputs** (<https://github.com/nipy/nipype/pull/1176>) (<https://github.com/nipy/nipype/issues/1175>)
- ENH: dipy.TrackDensityMap interface now accepts a reference image (<https://github.com/nipy/nipype/pull/1091>)
- FIX: Bug in XFibres5 (<https://github.com/nipy/nipype/pull/1168>)
- **ENH: Attempt to use hard links for data sink.** (<https://github.com/nipy/nipype/pull/1161>)
- **FIX: Updates to SGE Plugins** (<https://github.com/nipy/nipype/pull/1129>)
- **ENH: Add ants JointFusion() node with testing** (<https://github.com/nipy/nipype/pull/1160>)
- **ENH: Add –float option for antsRegistration calls** (<https://github.com/nipy/nipype/pull/1159>)
- **ENH: Added interface to simulate DWIs using the multi-tensor model** (<https://github.com/nipy/nipype/pull/1085>)
- ENH: New interface for FSL fslepgeom utility (<https://github.com/nipy/nipype/pull/1152>)
- ENH: Added SLURMGraph plugin for submitting jobs to SLURM with dependencies (<https://github.com/nipy/nipype/pull/1152>)

- [nipyre/pull/1136](#))
- **FIX: Enable absolute path definitions in DCMStack** (<https://github.com/nipy/nipyre/pull/1089>, replaced by <https://github.com/nipy/nipyre/pull/1093>)
 - **ENH: New mesh.MeshWarpMaths to operate on surface-defined warpings** (<https://github.com/nipy/nipyre/pull/1016>)
 - **FIX: Refactor P2PDistance, change name to ComputeMeshWarp, add regression tests, fix bug in area weighted distance, and added optimizations** (<https://github.com/nipy/nipyre/pull/1016>)
 - **ENH: Add an option not to resubmit Nodes that finished running when using SGEGraph** (<https://github.com/nipy/nipyre/pull/1002>)
 - **FIX: FUGUE is now properly listing outputs.** (<https://github.com/nipy/nipyre/pull/978>)
 - **ENH: Improved FieldMap-Based (FMB) workflow for correction of susceptibility distortions in EPI seqs.** (<https://github.com/nipy/nipyre/pull/1019>)
 - **FIX: In the FSLXcommand _list_outputs function fixed for loop range** (<https://github.com/nipy/nipyre/pull/1071>)
 - **ENH: Dropped support for now 7 years old Python 2.6** (<https://github.com/nipy/nipyre/pull/1069>)
 - **FIX: terminal_output is not mandatory anymore** (<https://github.com/nipy/nipyre/pull/1070>)
 - **ENH: Added “nipyre_cmd” tool for running interfaces from the command line** (<https://github.com/nipy/nipyre/pull/795>)
 - **FIX: Fixed Camino output naming** (<https://github.com/nipy/nipyre/pull/1061>)
 - **ENH: Add the average distance to ErrorMap** (<https://github.com/nipy/nipyre/pull/1039>)
 - **ENH: Inputs with name_source can be now chained in cascade** (<https://github.com/nipy/nipyre/pull/938>)
 - **ENH: Improve JSON interfaces: default settings when reading and consistent output creation when writing** (<https://github.com/nipy/nipyre/pull/1047>)
 - **FIX: AddCSVRow problems when using infields** (<https://github.com/nipy/nipyre/pull/1028>)
 - **FIX: Removed unused ANTS registration flag** (<https://github.com/nipy/nipyre/pull/999>)
 - **FIX: Amend create_tbss_non_fa() workflow to match FSL’s tbss_non_fa command.** (<https://github.com/nipy/nipyre/pull/1033>)
 - **FIX: remove unused mandatory flag from spm normalize** (<https://github.com/nipy/nipyre/pull/1048>)
 - **ENH: Update ANTSCorticalThickness interface** (<https://github.com/nipy/nipyre/pull/1013>)
 - **FIX: Edge case with sparsemodels and PEP8 cleanup** (<https://github.com/nipy/nipyre/pull/1046>)
 - **ENH: New io interfaces for JSON files reading/writing** (<https://github.com/nipy/nipyre/pull/1020>)
 - **ENH: Enhanced openfMRI script to support freesurfer linkage** (<https://github.com/nipy/nipyre/pull/1037>)
 - **BUG: matplotlib is supposed to be optional** (<https://github.com/nipy/nipyre/pull/1003>)
 - **FIX: Fix split_filename behaviour when path has no file component** (<https://github.com/nipy/nipyre/pull/1035>)
 - **ENH: Updated FSL dtifit to include option for grad non-linearities** (<https://github.com/nipy/nipyre/pull/1032>)
 - **ENH: Updated Camino tracking interfaces, which can now use FSL bedpostx output.** New options also include choice of tracker, interpolator, stepsize and curveinterval for angle threshold (<https://github.com/nipy/nipyre/pull/1029>)
 - **FIX: Interfaces redirecting X crashed if \$DISPLAY not defined** (<https://github.com/nipy/nipyre/pull/1027>)
 - **FIX: Bug crashed ‘make api’** (<https://github.com/nipy/nipyre/pull/1026>)
 - **ENH: Updated antsIntroduction to handle RA and RI registrations** (<https://github.com/nipy/nipyre/pull/1009>)
 - **ENH: Updated N4BiasCorrection input spec to include weight image and spline order. Made argument formatting consistent. Cleaned ants.segmentation according to PEP8.** (<https://github.com/nipy/nipyre/pull/990/files>)
 - **ENH: SPM12 Normalize interface** (<https://github.com/nipy/nipyre/pull/986>)
 - **FIX: Utility interface test dir** (<https://github.com/nipy/nipyre/pull/986>)
 - **FIX: IPython engine directory reset after crash** (<https://github.com/nipy/nipyre/pull/987>)
 - **ENH: Resting state fMRI example with NiPy realignment and no SPM** (<https://github.com/nipy/nipyre/pull/992>)
 - **FIX: Corrected Freesurfer SegStats _list_outputs to avoid error if summary_file is undefined** (issue #994)(<https://github.com/nipy/nipyre/pull/996>)
 - **FIX: OpenfMRI support and FSL 5.0.7 changes** (<https://github.com/nipy/nipyre/pull/1006>)
 - **FIX: Output prefix in SPM Normalize with modulation** (<https://github.com/nipy/nipyre/pull/1023>)
 - **ENH: Usability improvements in cluster environments** (<https://github.com/nipy/nipyre/pull/1025>)

- ENH: ANTs JointFusion() (<https://github.com/nipy/nipy/pull/1042>)
- ENH: Added csvReader() utility (<https://github.com/nipy/nipy/pull/1044>)
- FIX: typo in nipy.interfaces.freesurfer.utils.py Tkregister2 (<https://github.com/nipy/nipy/pull/1083>)
- FIX: SSHDataGrabber outputs now return full path to the grabbed/downloaded files. (<https://github.com/nipy/nipy/pull/1086>)
- FIX: Add QA output for TSNR to resting workflow (<https://github.com/nipy/nipy/pull/1088>)
- FIX: Change N4BiasFieldCorrection to use short tag for dimensionality (backward compatible) (<https://github.com/nipy/nipy/pull/1096>)
- ENH: Added -newgrid input to Warp in AFNI (3dWarp wrapper) (<https://github.com/nipy/nipy/pull/1128>)
- FIX: Fixed AFNI Copy interface to use positional inputs as required (<https://github.com/nipy/nipy/pull/1131>)
- ENH: Added a check in Dcm2nii to check if nipy created the config.ini file and remove if true (<https://github.com/nipy/nipy/pull/1132>)
- **ENH: Use a while loop to wait for Xvfb (up to a max wait time “xvfb_max_wait” in config file, default 10)** (<https://github.com/nipy/nipy/pull/1142>)

Release 0.10.0 (October 10, 2014)

- **ENH: New miscellaneous interfaces: SplitROIs (mapper), MergeROIs (reducer)** to enable parallel processing of very large images.
- **ENH: Updated FSL interfaces: BEDPOSTX and XFibres, former interfaces are still** available with the version suffix: BEDPOSTX4 and XFibres4. Added gpu versions of BEDPOSTX: BEDPOSTXGPU, BEDPOSTX5GPU, and BEDPOSTX4GPU
- ENH: Added experimental support for MIPAV algorithms thorough JIST plugins
- ENH: New dipy interfaces: Denoise, Resample
- ENH: New Freesurfer interfaces: Tkregister2 (for conversion of fsl style matrices to freesurfer format), MRIPretess
- ENH: New FSL interfaces: WarpPoints, WarpPointsToStd, EpiReg, ProbTrackX2, WarpUtils, ConvertWarp
- ENH: New miscellaneous interfaces: AddCSVRow, NormalizeProbabilityMapSet, AddNoise
- ENH: New AFNI interfaces: Eval, Means, SVMTest, SVMTrain
- **ENH: FUGUE interface has been refactored to use the name_template system, 3 examples** added to doctests, some bugs solved.
- **API: Interfaces to external packages are no longer available in the top-level nipy namespace,** and must be imported directly (e.g. from `nipy.interfaces import fsl`).
- **ENH: Support for elastix via a set of new interfaces: Registration, ApplyWarp, AnalyzeWarp,** PointsWarp, and EditTransform
- ENH: New ANTs interface: ApplyTransformsToPoints, LaplacianThickness
- ENH: New Diffusion Toolkit interface: TrackMerge
- ENH: New MRtrix interface: FilterTracks
- **ENH: New metrics group in algorithms. Now Distance, Overlap, and FuzzyOverlap** are found in `nipy.algorithms.metrics` instead of `misc`. Overlap interface extended to allow files containing multiple ROIs and volume physical units.
- ENH: New interface in `algorithms.metrics`: ErrorMap (a voxel-wise diff map).
- ENH: New FreeSurfer workflow: `create_skullstripped_recon_flow()`
- **ENH: Deep revision of workflows for correction of dMRI artifacts. New dmri_preprocessing** example.
- ENH: New data grabbing interface that works over SSH connections, SSHDataGrabber
- ENH: New color mode for `write_graph`
- ENH: You can now force MapNodes to be run serially
- ENH: Added ANTS based openfmri workflow
- ENH: MapNode now supports flattening of nested lists
- ENH: Support for headless mode using Xvfb
- ENH: `nipy_display_crash` has a debugging mode
- FIX: MRtrix tracking algorithms were ignoring mask parameters.
- FIX: FNIIRT registration pathway and associated OpenFMRI example script

- FIX: spm12b compatibility for Model estimate
- FIX: Batch scheduler controls the number of maximum jobs properly
- FIX: Update for FSL 5.0.7 which deprecated Contrast Manager

Release 0.9.2 (January 31, 2014)

- FIX: DataFinder was broken due to a typo
- FIX: Order of DataFinder outputs was not guaranteed, it's human sorted now
- ENH: New interfaces: Vnifti2Image, VtoMat

Release 0.9.1 (December 25, 2013)

- FIX: installation issues

Release 0.9.0 (December 20, 2013)

- ENH: SelectFiles: a streamlined version of DataGrabber
- ENH: new tools for defining workflows: JoinNode, synchronize and itersource
- ENH: W3C PROV support with optional RDF export built into Nipype
- ENH: Added support for Simple Linux Utility Resource Management (SLURM)
- **ENH: AFNI interfaces refactor, prefix, suffix are replaced by** “flexible_%s_templates”
- **ENH: New SPM interfaces:**
 - spm.ResliceToReference,
 - spm.DicomImport
- **ENH: New AFNI interfaces:**
 - afni.AFNItoNIFTI
 - afni.TCorr1D
- **ENH: Several new interfaces related to Camino were added:**
 - camino.SFPICOCalibData
 - camino.Conmat
 - camino.QBallMX
 - camino.LinRecon
 - camino.SFPeaks

One outdated interface no longer part of Camino was removed: - camino.Conmap
- **ENH: Three new mrtrix interfaces were added:**
 - mrtrix.GenerateDirections
 - mrtrix.FindShPeaks
 - mrtrix.Directions2Amplitude
- **ENH: New FSL interfaces:**
 - fsl.PrepareFieldmap
 - fsl.TOPUP
 - fsl.ApplyTOPUP
 - fsl.Eddy
- **ENH: New misc interfaces:**
 - FuzzyOverlap,
 - P2PDistance
- ENH: New workflows: nipype.workflows.dmri.fsl.epi.[fieldmap_correction&topup_correction]
- ENH: Added simplified outputname generation for command line interfaces.
- ENH: Allow ants use a single mask image
- ENH: Create configuration option for parameterizing directories with hashes
- ENH: arrange nodes by topological sort with disconnected subgraphs
- ENH: uses the nidm iri namespace for uuids

- ENH: remove old reporting webpage
- ENH: Added support for Vagrant
- API: ‘name’ is now a positional argument for Workflow, Node, and MapNode constructors
- API: SPM now defaults to SPM8 or SPM12b job format
- API: DataGrabber and SelectFiles use human (or natural) sort now
- **FIX: Several fixes related to Camino interfaces:**
 - ProcStreamlines would ignore many arguments silently (target, waypoint, exclusion ROIS, etc.)
 - DTLUTGen would silently round the “step”, “snr” and “trace” parameters to integers
 - PicoPDFs would not accept more than one lookup table
 - PicoPDFs default pdf did not correspond to Camino default
 - Track input model names were outdated (and would generate an error)
 - Track numpds parameter could not be set for deterministic tractography
 - FA created output files with erroneous extension
- FIX: Deals properly with 3d files in SPM Realign
- FIX: SPM with MCR fixed
- FIX: Cleaned up input and output spec metadata
- FIX: example openfmri script now makes the contrast spec a hashed input
- FIX: FILMGLS compatibility with FSL 5.0.5
- FIX: Freesurfer recon-all resume now avoids setting inputs
- FIX: File removal from node respects file associations img/hdr/mat, BRIK/HEAD

Release 0.8.0 (May 8, 2013)

- **ENH: New interfaces:** `nipy.Trim`, `fsl.GLM`, `fsl.SigLoss`, `spm.VBMSegment`, `fsl.InvWarp`, `dipy.TensorMode`
- ENH: Allow control over terminal output for commandline interfaces
- ENH: Added preliminary support for generating Python code from Workflows.
- **ENH: New workflows for dMRI and fMRI pre-processing:** added motion artifact correction with rotation of the B-matrix, and susceptibility correction for EPI imaging using fieldmaps. Updated eddy_correct pipeline to support both dMRI and fMRI, and new parameters.
- ENH: Minor improvements to FSL’s FUGUE and FLIRT interfaces
- ENH: Added optional dilation of parcels in `cmk.Parcellate`
- ENH: Interpolation mode added to `afni.Resample`
- **ENH: Function interface can accept a list of strings containing import statements** that allow external functions to run without their imports defined in the function body
- ENH: Allow node configurations to override master configuration
- FIX: SpecifyModel works with 3D files correctly now.

Release 0.7.0 (Dec 18, 2012)

- ENH: Add basic support for LSF plugin.
- **ENH: New interfaces:** `ICC`, `Meshfix`, `ants.Register`, `C3dAffineTool`, `ants.JacobianDeterminant`, `afni.AutoTcorrelate`, `DcmStack`
- ENH: New workflows: ants template building (both using ‘ANTS’ and the new ‘antsRegistration’)
- **ENH: New examples: how to use ANTS template building workflows** (`smri_ants_build_tmeplate`), how to set SGE specific options (`smri_ants_build_template_new`)
- ENH: added `no_flatten` option to Merge
- ENH: added versioning option and checking to traits
- ENH: added deprecation metadata to traits
- ENH: Slicer interfaces were updated to version 4.1

Release 0.6.0 (Jun 30, 2012)

- API: display variable no longer encoded as inputs in commandline interfaces
- ENH: input hash not modified when environment DISPLAY is changed
- ENH: support for 3d files for TSNR calculation
- ENH: Preliminary support for graph submission with SGE, PBS and Soma Workflow
- **ENH: New interfaces: MySQLSink, nipy.Similarity, WatershedBEM, MRIsSmooth,**
NetworkBasedStatistic, Atropos, N4BiasFieldCorrection, ApplyTransforms, fs.MakeAverageSubject,
epidewarp.fsl, WarpTimeSeriesImageMultiTransform, AVScale, mri_ms_LDA
- ENH: simple interfaces for spm
- FIX: CompCor component calculation was erroneous
- FIX: filename generation for AFNI and PRELUDE
- FIX: improved slicer module autogeneration
- FIX: added missing options for BBRegressor
- FIX: functionality of remove_unnecessary_outputs cleaned up
- FIX: local hash check works with appropriate inputs
- FIX: Captures all stdout from commandline programs
- FIX: Afni outputs should inherit from TraitedSpec

Release 0.5.3 (Mar 23, 2012)

- FIX: SPM model generation when output units is in scans

Release 0.5.2 (Mar 14, 2012)

- API: Node now allows specifying node level configuration for SGE/PBS clusters
- API: Logging to file is disabled by default
- API: New location of log file -> .nipy/nipy.cfg
- ENH: Changing logging options via config works for distributed processing
- FIX: Unittests on debian (logging and ipython)

Release 0.5 (Mar 10, 2012)

- API: FSL defaults to Nifti when OUTPUTTYPE environment variable not found
- API: By default inputs are removed from Node working directory
- API: InterfaceResult class is now versioned and stores class type not instance
- API: Added FIRST interface
- **API: Added max_jobs paramter to plugin_args. limits the number of jobs** executing at any given point in time
- API: crashdump_dir is now a config execution option
- **API: new config execution options for controlling hash checking, execution and logging** behavior when running in distributed mode.
- API: Node/MapNode has new attribute that allows it to run on master thread.
- API: IPython plugin now invokes IPython 0.11 or greater
- API: Canned workflows are now all under a different package structure
- API: SpecifyModel event_info renamed to event_files
- **API: DataGrabber is always being rerun (unless overwrite is set to False on Node level)**
- **API: “stop_on_first_rerun” does not stop for DataGrabber (unless overwrite is set to True on Node level)**
- **API: Output prefix can be set for spm nodes (SliceTiming, Realign, Coregister, Normalize, Smooth)**
- ENH: Added fsl resting state workflow based on behzadi 2007 CompCorr method.
- ENH: TSNR node produces mean and std-dev maps; allows polynomial detrending
- ENH: IdentityNodes are removed prior to execution

- ENH: Added Michael Notter's beginner's guide
- ENH: Added engine support for status callback functions
- ENH: SPM create warped node
- ENH: All underlying interfaces (including python ones) are now optional
- ENH: Added imperative programming option with Nodes and caching
- ENH: Added debug mode to configuration
- ENH: Results can be stored and loaded without traits exceptions
- ENH: Added concurrent log handler for distributed writing to log file
- ENH: Reporting can be turned off using config
- ENH: Added stats files to FreeSurferOutput
- ENH: Support for Condor through qsub emulation
- **ENH: IdentityNode with iterable expansion takes place after remaining Identity** Node removal
- ENH: Crashfile display script added
- ENH: Added FmriRealign4d node wrapped from nipy
- ENH: Added TBSS workflows and examples
- ENH: Support for openfmri data processing
- ENH: Package version check
- FIX: Fixed spm preproc workflow to cater to multiple functional runs
- FIX: Workflow outputs displays nodes with empty outputs
- FIX: SUSAN workflow works without usans
- FIX: SGE fixed for reading custom templates
- FIX: warping in SPM realign, Dartel and interpolation parameters
- FIX: Fixed voxel size parameter in freesurfer mri_convert
- FIX: 4D images in spm coregister
- FIX: Works around matlab tty bug
- FIX: Overwriting connection raises exception
- **FIX: Outputs are loaded from results and not stored in memory for during** distributed operation
- FIX: SPM threshold uses SPM.mat name and improved error detection
- FIX: Removing directory contents works even when a node has no outputs
- FIX: DARTEL workflows will run only when SPM 8 is available
- FIX: SPM Normalize estimate field fixed
- FIX: hashmethod argument now used for calculating hash of old file
- FIX: Modelgen now allows FSL style event files

Release 0.4.1 (Jun 16, 2011)

- Minor bugfixes

Release 0.4 (Jun 11, 2011)

- **API: Timestamp hashing does not use ctime anymore. Please update your hashes by** running workflows with updatehash=True option NOTE: THIS IS THE DEFAULT CONFIG NOW, so unless you updatehash, workflows will rerun
- **API: Workflow run function no longer supports (inseries, createdironly).** Functions used in connect string must be pickleable
- API: SPM EstimateContrast: ignore_derivs replaced by use_derivs
- API: All interfaces: added new config option ignore_exception
- **API: SpecifModel no longer supports (concatenate_runs, output_specs).** high_pass_filter cutoff is mandatory (even if its set to np.inf). Additional interfaces SpecifySPMModel and SpecifySparseModel support other types of data.
- API: fsl.DTIFit input "save" is now called "save_tensor"
- **API: All inputs of IdentityInterfaces are mandatory by default. You can turn** this off by specifying mandatory_inputs=False to the constructor.

- API: fsl FILMGLS input “autocorr_estimate” is now called “autocorr_estimate_only”
- **API: fsl ContrastMgr now requires access to specific files (no longer accepts the result directory)**
- **API: freesurfer.GLMFit input “surf” is now a boolean with three corresponding inputs** – subject_id, hemi, and surf_geo
- ENH: All commandline interfaces display stdout and stderr
- ENH: All interfaces raise exceptions on error with an option to suppress
- **ENH: Supports a plugin interface for execution (current support for multiprocessing, IPython, SGE, PBS)**
- ENH: MapNode runs in parallel under IPython, SGE, MultiProc, PBS
- ENH: Optionally allows keeping only required outputs
- **ENH: New interface: utility.Rename to change the name of files, optionally** using python string-formatting with inputs or regular expressions matching
- ENH: New interface: freesurfer.ApplyMask (mri_mask)
- ENH: New FSL interface – SwapDimensions (fslswapdim)
- ENH: Sparse models allow regressor scaling and temporal derivatives
- **ENH: Added support for the component parts of FSL’s TBSS workflow (TBSSSkeleton and DistanceMap)**
- ENH: dcm2nii interface exposes bvals, bvecs, reoriented and cropped images
- **ENH: Added several higher-level interfaces to the fslmaths command:**
ChangeDataType, Threshold, MeanImage, IsotropicSmooth, ApplyMask, TemporalFilter DilateImage, ErodeImage, SpatialFilter, UnaryMaths, BinaryMaths, MultiImageMaths
- ENH: added support for networkx 1.4 and improved iterable expansion
- ENH: Replaced BEDPOSTX and EddyCurrent with nipy pipelines
- ENH: Ability to create a hierarchical dot file
- ENH: Improved debugging information for rerunning nodes
- ENH: Added ‘stop_on_first_rerun’ option
- ENH: Added support for Camino
- ENH: Added support for Camino2Trackvis
- ENH: Added support for Connectome Viewer
- BF: dcm2nii interface handles gzipped files correctly
- BF: FNIRT generates proper outputs
- BF: fsl.DTIFit now properly collects tensor volume
- BF: updatehash now removes old result hash file

Release 0.3.4 (Jan 12, 2011)

- API: hash values for float use a string conversion up to the 10th decimal place.
- API: Iterables in output path will always be generated as _var1_val1_var2_val2 pairs
- ENH: Added support to nipy: GLM fit, contrast estimation and calculating mask from EPI
- **ENH: Added support for manipulating surface files in Freesurfer:**
projecting volume images onto the surface
smoothing along the surface
transforming a surface image from one subject to another
using tksurfer to save pictures of the surface
- ENH: Added support for flash processing using FreeSurfer
- ENH: Added support for flirt matrix in BBRegister
- ENH: Added support for FSL convert_xfm
- ENH: hashes can be updated again without rerunning all nodes.
- ENH: Added multiple regression design for FSL
- ENH: Added SPM based Analyze to Nifti converter
- ENH: Added increased support for PyXNAT
- ENH: Added support for MCR-based binary version of SPM
- ENH: Added SPM node for calculating various threshold statistics

- ENH: Added distance and dissimilarity measurements
- BF: Diffusion toolkit gets installed
- **BF: Changed FNIRT interface to accept flexible lists (rather than 4-tuples)** on all options specific to different subsampling levels

Release 0.3.3 (Sep 16, 2010)

- API: subject_id in ModelSpec is now deprecated
- API: spm.Threshold - does not need mask, beta, RPV anymore - takes only one image (stat_image - mind the name change) - works with SPM2 SPM.mat - returns additional map - pre topological FDR
- ENH: Added support for Diffusion toolkit
- ENH: Added support for FSL slicer and overlay
- ENH: Added support for dcm2nii
- BF: DataSink properly handles lists of lists now
- BF: DataGrabber has option for raising Exception on getting empty lists
- BF: Traits logic for 'requires' metadata
- BF: allows workflows to be relocatable
- BF: nested workflows with connections don't raise connection not found error
- BF: multiple workflows with identical nodenames and iterables do not create nested workflows

Release 0.3.2 (Aug 03, 2010)

Enhancements

- all outputs from nodes are now pickled as part of workflow processing
- added git developer docs

Bugs fixed

- FreeSurfer
- Fixed bugs in SegStats doctest

Release 0.3.1 (Jul 29, 2010)

Bugs fixed

- FreeSurfer
- Fixed bugs in glmfit and concatenate
- Added group t-test to freesurfer tutorial

Release 0.3 (Jul 27, 2010)

Incompatible changes

- Complete redesign of the Interface class - heavy use of Traits.
- Changes in the engine API - added Workflow and MapNode. Compulsory name argument.

Features added

- General:
- Type checking of inputs and outputs using Traits from [ETS](#).
- Support for nested workflows.

- Preliminary Slicer and AFNI support.
- New flexible DataGrabber node.
- AtlasPick and Threshold nodes.
- Preliminary support for XNAT.
- Doubled number of the tutorials.
- FSL:
- Added DTI processing nodes (note that TBSS nodes are still experimental).
- Recreated FEAT workflow.
- SPM:
- Added New Segment and many other nodes.
- Redesigned second level analysis.
- Developer

Release 0.13.1
Date May 21, 2017, 03:41 PDT

caching.memory

Module: `caching.memory`

Inheritance diagram for `nipytype.caching.memory`:

memory.PipeFunc

memory.Memory

Using nipy with persistence and lazy recomputation but without explicit name-steps pipeline: getting back scope in command-line based programming.
Change directory to provide relative paths for doctests >>> import os >>> filepath = os.path.dirname(os.path.realpath(__file__)) >>> datadir = os.path.realpath(os.path.join(filepath, '../testing/data')) >>> os.chdir(datadir)

Classes

Memory

class nipy.caching.memory.**Memory**(base_dir)

Bases: object

Memory context to provide caching for interfaces

Parameters base_dir: string :

The directory name of the location for the caching

Methods

<i>cache</i> (interface)	Returns a callable that caches the output of an interface
<i>clear_previous_runs</i> ([warn])	Remove all the cache that where not used in the latest run of the memory object: i.e.
<i>clear_previous_runs</i> ([warn])	Remove all the cache that where not used in the latest run of the memory object: i.e.

__init__(base_dir)

cache(interface)

Returns a callable that caches the output of an interface

Parameters interface: nipyte interface :

The nipyte interface class to be wrapped and cached

Returns pipe_func: a PipeFunc callable object :

An object that can be used as a function to apply the interface to arguments. Inputs of the interface are given as keyword arguments, bearing the same name as the name in the inputs specs of the interface.

Examples

```
>>> from tempfile import mkdtemp
>>> mem = Memory(mkdtemp())
>>> from nipyte.interfaces import fsl
```

Here we create a callable that can be used to apply an fsl.Merge interface to files

```
>>> fsl_merge = mem.cache(fsl.Merge)
```

Now we apply it to a list of files. We need to specify the list of input files and the dimension along which the files should be merged.

```
>>> results = fsl_merge(in_files=['a.nii', 'b.nii'],
...                      dimension='t')
```

We can retrieve the resulting file from the outputs: >>> results.outputs.merged_file # doctest: +SKIP
'...'

clear_previous_runs (warn=True)

Remove all the cache that where not used in the latest run of the memory object: i.e. since the corresponding Python object was created.

Parameters warn: boolean, optional :

If true, echoes warning messages for all directory removed

clear_runs_since (day=None, month=None, year=None, warn=True)

Remove all the cache that where not used since the given date

Parameters day, month, year: integers, optional :

The integers specifying the latest day (in localtime) that a node should have been accessed to be kept. If not given, the current date is used.

warn: boolean, optional :

If true, echoes warning messages for all directory removed

PipeFunc

```
class nipyte.caching.memory.PipeFunc (interface, base_dir, callback=None)
```

Bases: object

Callable interface to nipyte.interface objects

Use this to wrap nipyte.interface object and call them specifying their input with keyword arguments:

```
fsl_merge = PipeFunc(fsl.Merge, base_dir='.')
out = fsl_merge(in_files=files, dimension='t')
```

Methods

```
__call__(**kwargs)
```

```
__init__(interface, base_dir, callback=None)
```

Parameters interface: a nipyte interface class :

The interface class to wrap
base_dir: a string :
The directory in which the computation will be stored
callback: a callable :
An optional callable called each time after the function is called.

Functions

`nipyype.caching.memory.read_log(filename, run_dict=None)`
`nipyype.caching.memory.rm_all_but(base_dir, dirs_to_keep, warn=False)`
Remove all the sub-directories of base_dir, but those listed
Parameters base_dir: string :
The base directory
dirs_to_keep: set :
The names of the directories to keep

interfaces.base

Module: interfaces.base

Inheritance diagram for `nipyype.interfaces.base`:



Package contains interfaces for using existing functionality in other packages
Exaples FSL, matlab/SPM , afni
Requires Packages to be installed

Classes

BaseInterface

class `nipyype.interfaces.base.BaseInterface` (*from_file=None, **inputs*)
Bases: `nipyype.interfaces.base.Interface`
Implements common interface functionality.

Attributes

`output_spec`

Methods

<code>aggregate_outputs([runtime, needed_outputs])</code>	Collate expected outputs and check for existence
<code>help([returnhelp])</code>	Prints class help
<code>input_spec</code>	alias of <code>BaseInterfaceInputSpec</code>
<code>load_inputs_from_json(json_file[, overwrite])</code>	A convenient way to load pre-set inputs from a JSON file.
<code>run(**inputs)</code>	Execute this interface.
<code>save_inputs_to_json(json_file)</code>	A convenient way to save current inputs to a JSON file.

```

__init__(from_file=None, **inputs)
aggregate_outputs(runtime=None, needed_outputs=None)
    Collate expected outputs and check for existence
always_run
can_resume
classmethod help(returnhelp=False)
    Prints class help
input_spec
    alias of BaseInterfaceInputSpec
load_inputs_from_json(json_file, overwrite=True)
    A convenient way to load pre-set inputs from a JSON file.
output_spec = None
references_ = []
run(**inputs)
    Execute this interface.
    This interface will not raise an exception if runtime.returncode is non-zero.
    Parameters inputs : allows the interface settings to be updated
    Returns results : an InterfaceResult object containing a copy of the instance
        that was executed, provenance information and, if successful, results :
save_inputs_to_json(json_file)
    A convenient way to save current inputs to a JSON file.
version

```

BaseInterfaceInputSpec

```

class nipyte.interfaces.base.BaseInterfaceInputSpec(**kwargs)
    Bases: nipyte.interfaces.base.TraitedSpec

```

Methods

<code>add_class_trait(name, *trait)</code>	Adds a named trait attribute to this class.
<code>add_trait(name, *trait)</code>	Adds a trait attribute to this object.
<code>add_trait_category(category)</code>	Adds a trait category to a class.
<code>add_trait_listener(object[, prefix])</code>	
<code>all_trait_names()</code>	Returns the list of all trait names, including implicitly defined traits.
<code>base_trait(name)</code>	Returns the base trait definition for a trait attribute.
<code>class_default_traits_view()</code>	Returns the name of the default traits view for the class.

Continued on next page

Table 3.5 – continued from previous page

<code>class_editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<code>class_trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>class_trait_view([name, view_element])</code>	
<code>class_trait_view_elements()</code>	Returns the ViewElements object associated with the class.
<code>class_traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
<code>class_visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event
<code>clone_traits([traits, memo, copy])</code>	Clones a new object from this one, optionally copying only a specified set of traits.
<code>configure_traits([filename, view, kind, ...])</code>	Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
<code>copy_traits(other[, traits, memo, copy])</code>	Copies another object's trait attributes into this one.
<code>copyable_trait_names(**metadata)</code>	Returns the list of trait names to copy or clone by default.
<code>default_traits_view()</code>	Returns the name of the default traits view for the object's class.
<code>edit_traits([view, parent, kind, context, ...])</code>	Displays a user interface window for editing trait attribute values.
<code>editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.
<code>get(**kwargs)</code>	Returns traitled class as a dict
<code>get_hashval([hash_method])</code>	Return a dictionary of our items with hashes for each file.
<code>get_traitsfree(**kwargs)</code>	Returns traitled class as a dict
<code>has_metadata(name, metadata[, value, recursive])</code>	Return has_metadata for the requested trait name in this
<code>has_traits_interface(*interfaces)</code>	Returns whether the object implements a specified traits interface.
<code>items()</code>	Name, trait generator for user modifiable traits
<code>on_trait_change(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>on_trait_event(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>print_traits([show_help])</code>	Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.
<code>remove_trait(name)</code>	Removes a trait attribute from this object.
<code>remove_trait_listener(object[, prefix])</code>	
<code>reset_traits([traits])</code>	Resets some or all of an object's trait attributes to their default values.

Continued on next page

Table 3.5 – continued from previous page

<code>set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>set_trait_dispatch_handler(name, klass[, ...])</code>	Sets a trait notification dispatch handler.
<code>sync_trait(trait_name, object[, alias, ...])</code>	Synchronizes the value of a trait attribute on this object with a trait attribute on another object.
<code>trait(name[, force, copy])</code>	Returns the trait definition for the <i>name</i> trait attribute.
<code>trait_context()</code>	Returns the default context to use for editing or configuring traits.
<code>trait_get(*names, **metadata)</code>	Shortcut for getting object trait attributes.
<code>trait_items_event(event_trait, name, items_event)</code>	
<code>trait_monitor(handler[, remove])</code>	Adds or removes the specified <i>handler</i> from the list of active monitors.
<code>trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>trait_property_changed(...)</code>	
<code>trait_set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>trait_setq(**traits)</code>	Shortcut for setting object trait attributes.
<code>trait_subclasses([all])</code>	Returns a list of the immediate (or all) subclasses of this class.
<code>trait_view([name, view_element])</code>	Gets or sets a ViewElement associated with an object’s class.
<code>trait_view_elements()</code>	Returns the ViewElements object associated with the object’s class.
<code>trait_views([klass])</code>	Returns a list of the names of all view elements associated with the current object’s class.
<code>traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of <i>metadata</i> criteria.
<code>traits_init()</code>	
<code>traits_inited([True])</code>	
<code>validate_trait(name, value)</code>	Validates whether a value is legal for a trait.
<code>visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event

__init__ (***kwargs*)

Initialize handlers and inputs

add_class_trait (*name*, **trait*)

Adds a named trait attribute to this class.

Parameters *name* : str

Name of the attribute to add.

***trait** :

A trait or a value that can be converted to a trait using Trait() Trait definition of the attribute. It can be a single value or a list equivalent to an argument list for the Trait() function.

add_trait (*name*, **trait*)

Adds a trait attribute to this object.

Parameters *name* : str

Name of the attribute to add.

***trait** :

Trait or a value that can be converted to a trait by Trait(). Trait definition for *name*. If more than one value is specified, it is equivalent to passing the entire list of values to

Trait().

add_trait_category (*category*)
Adds a trait category to a class.

add_trait_listener (*object*, *prefix*='')
all_trait_names ()
Returns the list of all trait names, including implicitly defined traits.

base_trait (*name*)
Returns the base trait definition for a trait attribute.
Parameters **name** : str
Name of the attribute whose trait definition is returned.

class_default_traits_view ()
Returns the name of the default traits view for the class.

class_editable_traits ()
Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.

class_trait_names (***metadata*)
Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.
Parameters ****metadata** : :
Criteria for selecting trait attributes.

class_trait_view (*name*=None, *view_element*=None)
class_trait_view_elements ()
Returns the ViewElements object associated with the class.
The returned object can be used to access all the view elements associated with the class.

class_traits (***metadata*)
Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of *metadata* criteria.
Parameters ****metadata** : :
Criteria for selecting trait attributes.

class_visible_traits ()
Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class, that should be GUI visible

clone_traits (*traits*=None, *memo*=None, *copy*=None, ***metadata*)
Clones a new object from this one, optionally copying only a specified set of traits.
Parameters **traits** : list of strings
The list of names of the trait attributes to copy.
memo : dict
A dictionary of objects that have already been copied.
copy : str
The type of copy `deep` or `shallow` to perform on any trait that does not have explicit 'copy' metadata. A value of None means 'copy reference'.
Returns **new** : :
The newly cloned object.

configure_traits (*filename*=None, *view*=None, *kind*=None, *edit*=True, *context*=None, *handler*=None, *id*='', *scrollable*=None, ***args*)
Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
Parameters **filename** : str
The name (including path) of a file that contains a pickled representation of the current object. When this parameter is specified, the method reads the corresponding file (if it exists) to restore the saved values of the object's traits before displaying them. If the user confirms the dialog box (by clicking **OK**), the new values are written to the file. If this parameter is not specified, the values are loaded from the in-memory object, and are not persisted when the dialog box is closed.

view : View or str

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or None, the **kind** attribute of the View object is used.

edit : bool

Indicates whether to display a user interface. If *filename* specifies an existing file, setting *edit* to False loads the saved values from that file into the object without requiring user interaction.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns True on success. :

copy_traits (*other*, *traits=None*, *memo=None*, *copy=None*, ***metadata*)

Copies another object's trait attributes into this one.

Parameters other : object

The object whose trait attribute values should be copied.

traits : list of strings

A list of names of trait attributes to copy. If None or unspecified, the set of names returned by `trait_names()` is used. If 'all' or an empty list, the set of names returned by `all_trait_names()` is used.

memo : dict

A dictionary of objects that have already been copied.

copy : None | 'deep' | 'shallow'

The type of copy to perform on any trait that does not have explicit 'copy' metadata. A value of None means 'copy reference'.

Returns unassignable : list of strings

A list of attributes that the method was unable to copy, which is empty if all the attributes were successfully copied.

copyable_trait_names (***metadata*)

Returns the list of trait names to copy or clone by default.

default_traits_view ()

Returns the name of the default traits view for the object's class.

edit_traits (*view=None*, *parent=None*, *kind=None*, *context=None*, *handler=None*, *id=''*, *scrollable=None*, ***args*)

Displays a user interface window for editing trait attribute values.

Parameters view : View or string

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is

not specified, the View object returned by `trait_view()` is used.

parent : toolkit control

The reference to a user interface component to use as the parent window for the object's UI window.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or None, the **kind** attribute of the View object is used.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used.

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns A UI object. :

editable_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

get (**kwargs)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without notification handles

get_hashval (hash_method=None)

Return a dictionary of our items with hashes for each file.

Searches through dictionary items and if an item is a file, it calculates the md5 hash of the file contents and stores the file name and hash value as the new key value.

However, the overall bunch hash is calculated only on the hash value of a file. The path and name of the file are not used in the overall hash calculation.

Returns dict_withhash : dict

Copy of our dictionary with the new file hashes included with each file.

hashvalue : str

The md5 hash value of the traitled spec

get_traitsfree (**kwargs)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without any traits. The dictionary does not contain any attributes that were Undefined

has_metadata (name, metadata, value=None, recursive=True)

Return has_metadata for the requested trait name in this interface

has_traits_interface (*interfaces)

Returns whether the object implements a specified traits interface.

Parameters *interfaces : :

One or more traits Interface (sub)classes.

items ()

Name, trait generator for user modifiable traits

on_trait_change (handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters handler : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

on_trait_event (*handler*, *name=None*, *remove=False*, *dispatch='same'*, *priority=False*, *deferred=False*, *target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters handler : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

print_traits (*show_help=False*, ***metadata*)

Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.

Parameters show_help : bool

Indicates whether to display additional descriptive information.

remove_trait (*name*)

Removes a trait attribute from this object.

Parameters name : str

Name of the attribute to remove.

Returns result : bool

True if the trait was successfully removed.

remove_trait_listener (*object*, *prefix=''*)

reset_traits (*traits=None, **metadata*)
Resets some or all of an object’s trait attributes to their default values.
Parameters **traits** : list of strings
Names of trait attributes to reset.
Returns **unresetable** : list of strings
A list of attributes that the method was unable to reset, which is empty if all the attributes were successfully reset.

set (*trait_change_notify=True, **traits*)
Shortcut for setting object trait attributes.
Parameters **trait_change_notify** : bool
If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: `trait_setq`)
****traits** :
Key/value pairs, the trait attributes and their values to be set
Returns **self** :
The method returns this object, after setting attributes.

set_trait_dispatch_handler (*name, klass, override=False*)
Sets a trait notification dispatch handler.

sync_trait (*trait_name, object, alias=None, mutual=True, remove=False*)
Synchronizes the value of a trait attribute on this object with a trait attribute on another object.
Parameters **name** : str
Name of the trait attribute on this object.
object : object
The object with which to synchronize.
alias : str
Name of the trait attribute on *other*; if None or omitted, same as *name*.
mutual : bool or int
Indicates whether synchronization is mutual (True or non-zero) or one-way (False or zero)
remove : bool or int
Indicates whether synchronization is being added (False or zero) or removed (True or non-zero)

trait (*name, force=False, copy=False*)
Returns the trait definition for the *name* trait attribute.
Parameters **name** : str
Name of the attribute whose trait definition is to be returned.
force : bool
Indicates whether to return a trait definition if *name* is not explicitly defined.
copy : bool
Indicates whether to return the original trait definition or a copy.

trait_context ()
Returns the default context to use for editing or configuring traits.

trait_get (**names, **metadata*)
Shortcut for getting object trait attributes.
Parameters **names** : list of strings
A list of trait attribute names whose values are requested.
Returns **result** : dict
A dictionary whose keys are the names passed as arguments and whose values are the corresponding trait values.

trait_items_event (*event_trait, name, items_event*)

trait_monitor (*handler, remove=False*)
Adds or removes the specified *handler* from the list of active monitors.
Parameters **handler** : function
The function to add or remove as a monitor.

remove : bool
Flag indicating whether to remove (True) or add the specified handler as a monitor for this class.

trait_names (***metadata*)
Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters ***metadata* : :
Criteria for selecting trait attributes.

trait_property_changed (*name*, *old_value* [, *new_value*])
trait_set (*trait_change_notify*=True, ***traits*)
Shortcut for setting object trait attributes.

Parameters **trait_change_notify** : bool
If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: *trait_setq*)

****traits** : :
Key/value pairs, the trait attributes and their values to be set

Returns **self** : :
The method returns this object, after setting attributes.

trait_setq (***traits*)
Shortcut for setting object trait attributes.

Parameters ***traits* : :
Key/value pairs, the trait attributes and their values to be set. No trait change notifications will be generated for any values assigned (see also: *trait_set*).

Returns **self** : :
The method returns this object, after setting attributes.

trait_subclasses (*all*=False)
Returns a list of the immediate (or all) subclasses of this class.

Parameters **all** : bool
Indicates whether to return all subclasses of this class. If False, only immediate subclasses are returned.

trait_view (*name*=None, *view_element*=None)
Gets or sets a ViewElement associated with an object's class.

Parameters **name** : str
Name of a view element

view_element : ViewElement
View element to associate

Returns **A view element.** :
A ViewElement object

trait_view_elements ()
Returns the ViewElements object associated with the object's class.
The returned object can be used to access all the view elements associated with the class.

trait_views (*klass*=None)
Returns a list of the names of all view elements associated with the current object's class.

Parameters **klass** : class
A class, such that all returned names must correspond to instances of this class. Possible values include:
Group
Item
View
ViewElement
ViewSubElement

traits (***metadata*)
Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of *metadata* criteria.

Parameters ***metadata* : :

Criteria for selecting trait attributes.

traits_init()

traits_inited([True])

validate_trait(name, value)

Validates whether a value is legal for a trait.

Returns the validated value if it is valid.

visible_traits()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object, that should be GUI visible

wrappers = {'ui': <class 'traits.trait_notifiers.FastUITraitChangeNotifyWrapper'>, 'new': <class 'traits.trait_notif

BaseTraitedSpec

class nipy.interfaces.base.**BaseTraitedSpec** (**kwargs)

Bases: traits.has_traits.HasTraits

Provide a few methods necessary to support nipy interface api

The inputs attribute of interfaces call certain methods that are not available in traits.HasTraits. These are provided here.

new metadata:

usedefault : set this to True if the default value of the trait should be used. Unless this is set, the attributes are set to traits.Undefined

new attribute:

get_hashval : returns a tuple containing the state of the trait as a dict and hashvalue corresponding to dict. XXX Reconsider this in the long run, but it seems like the best solution to move forward on the refactoring.

Methods

<code>add_class_trait(name, *trait)</code>	Adds a named trait attribute to this class.
<code>add_trait(name, *trait)</code>	Adds a trait attribute to this object.
<code>add_trait_category(category)</code>	Adds a trait category to a class.
<code>add_trait_listener(object[, prefix])</code>	
<code>all_trait_names()</code>	Returns the list of all trait names, including implicitly defined traits.
<code>base_trait(name)</code>	Returns the base trait definition for a trait attribute.
<code>class_default_traits_view()</code>	Returns the name of the default traits view for the class.
<code>class_editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<code>class_trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>class_trait_view([name, view_element])</code>	
<code>class_trait_view_elements()</code>	Returns the ViewElements object associated with the class.
<code>class_traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
<code>class_visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event

Continued on next page

Table 3.6 – continued from previous page

<code>clone_traits([traits, memo, copy])</code>	Clones a new object from this one, optionally copying only a specified set of traits.
<code>configure_traits([filename, view, kind, ...])</code>	Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
<code>copy_traits(other[, traits, memo, copy])</code>	Copies another object's trait attributes into this one.
<code>copyable_trait_names(**metadata)</code>	Returns the list of trait names to copy or clone by default.
<code>default_traits_view()</code>	Returns the name of the default traits view for the object's class.
<code>edit_traits([view, parent, kind, context, ...])</code>	Displays a user interface window for editing trait attribute values.
<code>editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.
<code>get(**kwargs)</code>	Returns traitled class as a dict
<code>get_hashval([hash_method])</code>	Return a dictionary of our items with hashes for each file.
<code>get_traitsfree(**kwargs)</code>	Returns traitled class as a dict
<code>has_metadata(name, metadata[, value, recursive])</code>	Return has_metadata for the requested trait name in this
<code>has_traits_interface(*interfaces)</code>	Returns whether the object implements a specified traits interface.
<code>items()</code>	Name, trait generator for user modifiable traits
<code>on_trait_change(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>on_trait_event(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>print_traits([show_help])</code>	Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.
<code>remove_trait(name)</code>	Removes a trait attribute from this object.
<code>remove_trait_listener(object[, prefix])</code>	
<code>reset_traits([traits])</code>	Resets some or all of an object's trait attributes to their default values.
<code>set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>set_trait_dispatch_handler(name, klass[, ...])</code>	Sets a trait notification dispatch handler.
<code>sync_trait(trait_name, object[, alias, ...])</code>	Synchronizes the value of a trait attribute on this object with a trait attribute on another object.
<code>trait(name[, force, copy])</code>	Returns the trait definition for the <i>name</i> trait attribute.
<code>trait_context()</code>	Returns the default context to use for editing or configuring traits.
<code>trait_get(*names, **metadata)</code>	Shortcut for getting object trait attributes.
<code>trait_items_event(event_trait, name, items_event)</code>	
<code>trait_monitor(handler[, remove])</code>	Adds or removes the specified <i>handler</i> from the list of active monitors.

Continued on next page

Table 3.6 – continued from previous page

<code>trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>trait_property_changed(...)</code>	
<code>trait_set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>trait_setq(**traits)</code>	Shortcut for setting object trait attributes.
<code>trait_subclasses([all])</code>	Returns a list of the immediate (or all) subclasses of this class.
<code>trait_view([name, view_element])</code>	Gets or sets a ViewElement associated with an object’s class.
<code>trait_view_elements()</code>	Returns the ViewElements object associated with the object’s class.
<code>trait_views([klass])</code>	Returns a list of the names of all view elements associated with the current object’s class.
<code>traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of <i>metadata</i> criteria.
<code>traits_init()</code>	
<code>traits_inited([True])</code>	
<code>validate_trait(name, value)</code>	Validates whether a value is legal for a trait.
<code>visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event

__init__ (**kwargs)

Initialize handlers and inputs

add_class_trait (name, *trait)

Adds a named trait attribute to this class.

Parameters name : str

Name of the attribute to add.

*trait : :

A trait or a value that can be converted to a trait using Trait() Trait definition of the attribute. It can be a single value or a list equivalent to an argument list for the Trait() function.

add_trait (name, *trait)

Adds a trait attribute to this object.

Parameters name : str

Name of the attribute to add.

*trait : :

Trait or a value that can be converted to a trait by Trait(). Trait definition for name. If more than one value is specified, it is equivalent to passing the entire list of values to Trait().

add_trait_category (category)

Adds a trait category to a class.

add_trait_listener (object, prefix='')

all_trait_names ()

Returns the list of all trait names, including implicitly defined traits.

base_trait (name)

Returns the base trait definition for a trait attribute.

Parameters name : str

Name of the attribute whose trait definition is returned.

class_default_traits_view ()

Returns the name of the default traits view for the class.

class_editable_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.

class_trait_names (***metadata*)

Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters ***metadata* :

Criteria for selecting trait attributes.

class_trait_view (*name=None*, *view_element=None*)

class_trait_view_elements ()

Returns the ViewElements object associated with the class.

The returned object can be used to access all the view elements associated with the class.

class_traits (***metadata*)

Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of *metadata* criteria.

Parameters ***metadata* :

Criteria for selecting trait attributes.

class_visible_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class, that should be GUI visible

clone_traits (*traits=None*, *memo=None*, *copy=None*, ***metadata*)

Clones a new object from this one, optionally copying only a specified set of traits.

Parameters *traits* : list of strings

The list of names of the trait attributes to copy.

memo : dict

A dictionary of objects that have already been copied.

copy : str

The type of copy *deep* or *shallow* to perform on any trait that does not have explicit 'copy' metadata. A value of *None* means 'copy reference'.

Returns *new* :

The newly cloned object.

configure_traits (*filename=None*, *view=None*, *kind=None*, *edit=True*, *context=None*, *handler=None*, *id=''*, *scrollable=None*, ***args*)

Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.

Parameters *filename* : str

The name (including path) of a file that contains a pickled representation of the current object. When this parameter is specified, the method reads the corresponding file (if it exists) to restore the saved values of the object's traits before displaying them. If the user confirms the dialog box (by clicking **OK**), the new values are written to the file. If this parameter is not specified, the values are loaded from the in-memory object, and are not persisted when the dialog box is closed.

view : View or str

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or *None*, the **kind** attribute of the View object is used.

edit : bool

Indicates whether to display a user interface. If *filename* specifies an existing file, setting *edit* to *False* loads the saved values from that file into the object without requiring user interaction.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns True on success. :

copy_traits (*other, traits=None, memo=None, copy=None, **metadata*)

Copies another object's trait attributes into this one.

Parameters other : object

The object whose trait attribute values should be copied.

traits : list of strings

A list of names of trait attributes to copy. If None or unspecified, the set of names returned by `trait_names()` is used. If 'all' or an empty list, the set of names returned by `all_trait_names()` is used.

memo : dict

A dictionary of objects that have already been copied.

copy : None | 'deep' | 'shallow'

The type of copy to perform on any trait that does not have explicit 'copy' metadata. A value of None means 'copy reference'.

Returns unassignable : list of strings

A list of attributes that the method was unable to copy, which is empty if all the attributes were successfully copied.

copyable_trait_names (***metadata*)

Returns the list of trait names to copy or clone by default.

default_traits_view ()

Returns the name of the default traits view for the object's class.

edit_traits (*view=None, parent=None, kind=None, context=None, handler=None, id='', scrollable=None, **args*)

Displays a user interface window for editing trait attribute values.

Parameters view : View or string

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

parent : toolkit control

The reference to a user interface component to use as the parent window for the object's UI window.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or None, the **kind** attribute of the View object is used.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used.

handler : Handler

A handler object used for event handling in the dialog box. If None, the default

handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns A UI object. :

editable_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

get (**kwargs)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without notification handles

get_hashval (hash_method=None)

Return a dictionary of our items with hashes for each file.

Searches through dictionary items and if an item is a file, it calculates the md5 hash of the file contents and stores the file name and hash value as the new key value.

However, the overall bunch hash is calculated only on the hash value of a file. The path and name of the file are not used in the overall hash calculation.

Returns dict_withhash : dict

Copy of our dictionary with the new file hashes included with each file.

hashvalue : str

The md5 hash value of the traitled spec

get_traitsfree (**kwargs)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without any traits. The dictionary does not contain any attributes that were Undefined

has_metadata (name, metadata, value=None, recursive=True)

Return has_metadata for the requested trait name in this interface

has_traits_interface (*interfaces)

Returns whether the object implements a specified traits interface.

Parameters *interfaces : :

One or more traits Interface (sub)classes.

items ()

Name, trait generator for user modifiable traits

on_trait_change (handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters handler : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

on_trait_event (*handler*, *name=None*, *remove=False*, *dispatch='same'*, *priority=False*, *deferred=False*, *target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters *handler* : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

print_traits (*show_help=False*, ***metadata*)

Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.

Parameters *show_help* : bool

Indicates whether to display additional descriptive information.

remove_trait (*name*)

Removes a trait attribute from this object.

Parameters *name* : str

Name of the attribute to remove.

Returns *result* : bool

True if the trait was successfully removed.

remove_trait_listener (*object*, *prefix=''*)

reset_traits (*traits=None*, ***metadata*)

Resets some or all of an object's trait attributes to their default values.

Parameters *traits* : list of strings

Names of trait attributes to reset.

Returns *unresetable* : list of strings

A list of attributes that the method was unable to reset, which is empty if all the attributes were successfully reset.

set (*trait_change_notify=True*, ***traits*)

Shortcut for setting object trait attributes.

Parameters *trait_change_notify* : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: `trait_setq`)

****traits :**
Key/value pairs, the trait attributes and their values to be set

Returns self :
The method returns this object, after setting attributes.

set_trait_dispatch_handler (*name, klass, override=False*)
Sets a trait notification dispatch handler.

sync_trait (*trait_name, object, alias=None, mutual=True, remove=False*)
Synchronizes the value of a trait attribute on this object with a trait attribute on another object.

Parameters name : str
Name of the trait attribute on this object.

object : object
The object with which to synchronize.

alias : str
Name of the trait attribute on *other*; if None or omitted, same as *name*.

mutual : bool or int
Indicates whether synchronization is mutual (True or non-zero) or one-way (False or zero)

remove : bool or int
Indicates whether synchronization is being added (False or zero) or removed (True or non-zero)

trait (*name, force=False, copy=False*)
Returns the trait definition for the *name* trait attribute.

Parameters name : str
Name of the attribute whose trait definition is to be returned.

force : bool
Indicates whether to return a trait definition if *name* is not explicitly defined.

copy : bool
Indicates whether to return the original trait definition or a copy.

trait_context ()
Returns the default context to use for editing or configuring traits.

trait_get (**names, **metadata*)
Shortcut for getting object trait attributes.

Parameters names : list of strings
A list of trait attribute names whose values are requested.

Returns result : dict
A dictionary whose keys are the names passed as arguments and whose values are the corresponding trait values.

trait_items_event (*event_trait, name, items_event*)

trait_monitor (*handler, remove=False*)
Adds or removes the specified *handler* from the list of active monitors.

Parameters handler : function
The function to add or remove as a monitor.

remove : bool
Flag indicating whether to remove (True) or add the specified handler as a monitor for this class.

trait_names (***metadata*)
Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters **metadata :
Criteria for selecting trait attributes.

trait_property_changed (*name, old_value[, new_value]*)

trait_set (*trait_change_notify=True, **traits*)
Shortcut for setting object trait attributes.

Parameters trait_change_notify : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: `trait_setq`)

****traits :**
Key/value pairs, the trait attributes and their values to be set

Returns self :
The method returns this object, after setting attributes.

trait_setq (***traits*)
Shortcut for setting object trait attributes.

Parameters **traits :
Key/value pairs, the trait attributes and their values to be set. No trait change notifications will be generated for any values assigned (see also: `trait_set`).

Returns self :
The method returns this object, after setting attributes.

trait_subclasses (*all=False*)
Returns a list of the immediate (or all) subclasses of this class.

Parameters all : bool
Indicates whether to return all subclasses of this class. If False, only immediate subclasses are returned.

trait_view (*name=None, view_element=None*)
Gets or sets a ViewElement associated with an object's class.

Parameters name : str
Name of a view element

view_element : ViewElement
View element to associate

Returns A view element. :

trait_view_elements ()
Returns the ViewElements object associated with the object's class.
The returned object can be used to access all the view elements associated with the class.

trait_views (*klass=None*)
Returns a list of the names of all view elements associated with the current object's class.

Parameters klass : class
A class, such that all returned names must correspond to instances of this class. Possible values include:
Group
Item
View
ViewElement
ViewSubElement

traits (***metadata*)
Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of *metadata* criteria.

Parameters **metadata :
Criteria for selecting trait attributes.

traits_init ()

traits_inited ([*True*])

validate_trait (*name, value*)
Validates whether a value is legal for a trait.
Returns the validated value if it is valid.

visible_traits ()
Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object, that should be GUI visible

wrappers = {'ui': <class 'traits.trait_notifiers.FastUITraitChangeNotifyWrapper'>, 'new': <class 'traits.trait_notifi

Bunch

class `nipy.interfaces.base.Bunch` (*args, **kwargs)

Bases: `object`

Dictionary-like class that provides attribute-style access to it's items.

A *Bunch* is a simple container that stores it's items as class attributes. Internally all items are stored in a dictionary and the class exposes several of the dictionary methods.

Notes

The Bunch pattern came from the Python Cookbook:

Examples

```
>>> from nipy.interfaces.base import Bunch
>>> inputs = Bunch(infile='subj.nii', fwhm=6.0, register_to_mean=True)
>>> inputs
Bunch(fwhm=6.0, infile='subj.nii', register_to_mean=True)
>>> inputs.register_to_mean = False
>>> inputs
Bunch(fwhm=6.0, infile='subj.nii', register_to_mean=False)
```

Methods

<code>dictcopy()</code>	returns a deep copy of existing Bunch as a dictionary
<code>get(*args)</code>	Support dictionary <code>get()</code> functionality
<code>items()</code>	iterates over bunch attributes as key, value pairs
<code>iteritems()</code>	iterates over bunch attributes as key, value pairs
<code>set(**kwargs)</code>	Support dictionary <code>get()</code> functionality
<code>update(*args, **kwargs)</code>	update existing attribute, or create new attribute

`__init__` (*args, **kwargs)

`dictcopy` ()

returns a deep copy of existing Bunch as a dictionary

`get` (*args)

Support dictionary `get()` functionality

`items` ()

iterates over bunch attributes as key, value pairs

`iteritems` ()

iterates over bunch attributes as key, value pairs

`set` (**kwargs)

Support dictionary `get()` functionality

`update` (*args, **kwargs)

update existing attribute, or create new attribute

Note: update is very much like `HasTraits.set`

CommandLine

class `nipy.interfaces.base.CommandLine` (command=None, **inputs)

Bases: `nipy.interfaces.base.BaseInterface`

Implements functionality to interact with command line programs class must be instantiated with a command argument

Parameters `command` : string

define base immutable *command* you wish to run
args : string, optional
optional arguments passed to base *command*

Examples

```
>>> import pprint
>>> from nipyne.interfaces.base import CommandLine
>>> cli = CommandLine(command='ls', environ={'DISPLAY': ':1'})
>>> cli.inputs.args = '-al'
>>> cli.cmdline
'ls -al'
```

```
>>> pprint.pprint(cli.inputs.trait_get())
{'args': '-al',
 'environ': {'DISPLAY': ':1'},
 'ignore_exception': False,
 'terminal_output': 'stream'}
```

```
>>> cli.inputs.get_hashval()[0][0]
('args', '-al')
>>> cli.inputs.get_hashval()[1]
'11c37f97649cd61627f4afe5136af8c0'
```

Attributes

output_spec

Methods

<i>aggregate_outputs</i> (runtime, needed_outputs)	Collate expected outputs and check for existence
<i>help</i> ([returnhelp])	
<i>input_spec</i>	alias of <i>CommandLineInputSpec</i>
<i>load_inputs_from_json</i> (json_file[, over-write])	A convenient way to load pre-set inputs from a JSON file.
<i>raise_exception</i> (runtime)	
<i>run</i> (**inputs)	Execute this interface.
<i>save_inputs_to_json</i> (json_file)	A convenient way to save current inputs to a JSON file.
<i>set_default_terminal_output</i> (output_type)	Set the default terminal output for CommandLine Interfaces.
<i>version_from_command</i> ([flag])	

```
__init__ (command=None, **inputs)
aggregate_outputs (runtime=None, needed_outputs=None)
    Collate expected outputs and check for existence
always_run
can_resume
cmd
    sets base command, immutable
```


cmdline

command plus any arguments (args) validates arguments and generates command line

classmethod **help** (*returnhelp=False*)

input_spec

alias of *CommandLineInputSpec*

load_inputs_from_json (*json_file, overwrite=True*)

A convenient way to load pre-set inputs from a JSON file.

output_spec = None

raise_exception (*runtime*)

references_ = []

run (***inputs*)

Execute this interface.

This interface will not raise an exception if runtime.returncode is non-zero.

Parameters **inputs** : allows the interface settings to be updated

Returns **results** : an InterfaceResult object containing a copy of the instance

that was executed, provenance information and, if successful, results :

save_inputs_to_json (*json_file*)

A convenient way to save current inputs to a JSON file.

classmethod **set_default_terminal_output** (*output_type*)

Set the default terminal output for CommandLine Interfaces.

This method is used to set default terminal output for CommandLine Interfaces. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.terminal_output.

version

version_from_command (*flag='-v'*)

CommandLineInputSpec

class *nipytype.interfaces.base.CommandLineInputSpec* (***kwargs*)

Bases: *nipytype.interfaces.base.BaseInterfaceInputSpec*

Methods

<i>add_class_trait</i> (name, *trait)	Adds a named trait attribute to this class.
<i>add_trait</i> (name, *trait)	Adds a trait attribute to this object.
<i>add_trait_category</i> (category)	Adds a trait category to a class.
<i>add_trait_listener</i> (object[, prefix])	
<i>all_trait_names</i> ()	Returns the list of all trait names, including implicitly defined traits.
<i>base_trait</i> (name)	Returns the base trait definition for a trait attribute.
<i>class_default_traits_view</i> ()	Returns the name of the default traits view for the class.
<i>class_editable_traits</i> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<i>class_trait_names</i> (**metadata)	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<i>class_trait_view</i> ([name, view_element])	
<i>class_trait_view_elements</i> ()	Returns the ViewElements object associated with the class.

Continued on next page

Table 3.10 – continued from previous page

<code>class_traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
<code>class_visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event
<code>clone_traits([traits, memo, copy])</code>	Clones a new object from this one, optionally copying only a specified set of traits.
<code>configure_traits([filename, view, kind, ...])</code>	Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
<code>copy_traits(other[, traits, memo, copy])</code>	Copies another object's trait attributes into this one.
<code>copyable_trait_names(**metadata)</code>	Returns the list of trait names to copy or clone by default.
<code>default_traits_view()</code>	Returns the name of the default traits view for the object's class.
<code>edit_traits([view, parent, kind, context, ...])</code>	Displays a user interface window for editing trait attribute values.
<code>editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.
<code>get(**kwargs)</code>	Returns traitled class as a dict
<code>get_hashval([hash_method])</code>	Return a dictionary of our items with hashes for each file.
<code>get_traitsfree(**kwargs)</code>	Returns traitled class as a dict
<code>has_metadata(name, metadata[, value, recursive])</code>	Return has_metadata for the requested trait name in this
<code>has_traits_interface(*interfaces)</code>	Returns whether the object implements a specified traits interface.
<code>items()</code>	Name, trait generator for user modifiable traits
<code>on_trait_change(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>on_trait_event(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>print_traits([show_help])</code>	Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.
<code>remove_trait(name)</code>	Removes a trait attribute from this object.
<code>remove_trait_listener(object[, prefix])</code>	
<code>reset_traits([traits])</code>	Resets some or all of an object's trait attributes to their default values.
<code>set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>set_trait_dispatch_handler(name, klass[, ...])</code>	Sets a trait notification dispatch handler.
<code>sync_trait(trait_name, object[, alias, ...])</code>	Synchronizes the value of a trait attribute on this object with a trait attribute on another object.
<code>trait(name[, force, copy])</code>	Returns the trait definition for the <i>name</i> trait attribute.
<code>trait_context()</code>	Returns the default context to use for editing or configuring traits.
<code>trait_get(*names, **metadata)</code>	Shortcut for getting object trait attributes.

Continued on next page

Table 3.10 – continued from previous page

<code>trait_items_event(event_trait, name, items_event)</code>	
<code>trait_monitor(handler[, remove])</code>	Adds or removes the specified <i>handler</i> from the list of active monitors.
<code>trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>trait_property_changed(...)</code>	
<code>trait_set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>trait_setq(**traits)</code>	Shortcut for setting object trait attributes.
<code>trait_subclasses([all])</code>	Returns a list of the immediate (or all) subclasses of this class.
<code>trait_view([name, view_element])</code>	Gets or sets a ViewElement associated with an object’s class.
<code>trait_view_elements()</code>	Returns the ViewElements object associated with the object’s class.
<code>trait_views([klass])</code>	Returns a list of the names of all view elements associated with the current object’s class.
<code>traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of <i>metadata</i> criteria.
<code>traits_init()</code>	
<code>traits_inited([True])</code>	
<code>validate_trait(name, value)</code>	Validates whether a value is legal for a trait.
<code>visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event

__init__ (***kwargs*)

Initialize handlers and inputs

add_class_trait (*name*, **trait*)

Adds a named trait attribute to this class.

Parameters *name* : str

Name of the attribute to add.

***trait** :

A trait or a value that can be converted to a trait using Trait() Trait definition of the attribute. It can be a single value or a list equivalent to an argument list for the Trait() function.

add_trait (*name*, **trait*)

Adds a trait attribute to this object.

Parameters *name* : str

Name of the attribute to add.

***trait** :

Trait or a value that can be converted to a trait by Trait(). Trait definition for *name*. If more than one value is specified, it is equivalent to passing the entire list of values to Trait().

add_trait_category (*category*)

Adds a trait category to a class.

add_trait_listener (*object*, *prefix*=’')

all_trait_names ()

Returns the list of all trait names, including implicitly defined traits.

base_trait (*name*)

Returns the base trait definition for a trait attribute.

Parameters *name* : str

Name of the attribute whose trait definition is returned.

class_default_traits_view()

Returns the name of the default traits view for the class.

class_editable_traits()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.

class_trait_names(metadata)**

Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters ****metadata :**

Criteria for selecting trait attributes.

class_trait_view(name=None, view_element=None)

class_trait_view_elements()

Returns the ViewElements object associated with the class.

The returned object can be used to access all the view elements associated with the class.

class_traits(metadata)**

Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of *metadata* criteria.

Parameters ****metadata :**

Criteria for selecting trait attributes.

class_visible_traits()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class, that should be GUI visible

clone_traits(traits=None, memo=None, copy=None, **metadata)

Clones a new object from this one, optionally copying only a specified set of traits.

Parameters **traits :** list of strings

The list of names of the trait attributes to copy.

memo : dict

A dictionary of objects that have already been copied.

copy : str

The type of copy `deep` or `shallow` to perform on any trait that does not have explicit 'copy' metadata. A value of `None` means 'copy reference'.

Returns **new :**

The newly cloned object.

configure_traits(filename=None, view=None, kind=None, edit=True, context=None, handler=None, id='', scrollable=None, **args)

Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.

Parameters **filename :** str

The name (including path) of a file that contains a pickled representation of the current object. When this parameter is specified, the method reads the corresponding file (if it exists) to restore the saved values of the object's traits before displaying them. If the user confirms the dialog box (by clicking **OK**), the new values are written to the file. If this parameter is not specified, the values are loaded from the in-memory object, and are not persisted when the dialog box is closed.

view : View or str

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or `None`, the **kind** attribute of the View object is used.

edit : bool

Indicates whether to display a user interface. If *filename* specifies an existing file, setting *edit* to False loads the saved values from that file into the object without requiring user interaction.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns True on success. :

copy_traits (*other*, *traits=None*, *memo=None*, *copy=None*, ***metadata*)

Copies another object's trait attributes into this one.

Parameters other : object

The object whose trait attribute values should be copied.

traits : list of strings

A list of names of trait attributes to copy. If None or unspecified, the set of names returned by `trait_names()` is used. If 'all' or an empty list, the set of names returned by `all_trait_names()` is used.

memo : dict

A dictionary of objects that have already been copied.

copy : None | 'deep' | 'shallow'

The type of copy to perform on any trait that does not have explicit 'copy' metadata. A value of None means 'copy reference'.

Returns unassignable : list of strings

A list of attributes that the method was unable to copy, which is empty if all the attributes were successfully copied.

copyable_trait_names (***metadata*)

Returns the list of trait names to copy or clone by default.

default_traits_view ()

Returns the name of the default traits view for the object's class.

edit_traits (*view=None*, *parent=None*, *kind=None*, *context=None*, *handler=None*, *id=''*, *scrollable=None*, ***args*)

Displays a user interface window for editing trait attribute values.

Parameters view : View or string

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

parent : toolkit control

The reference to a user interface component to use as the parent window for the object's UI window.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or None, the **kind** attribute of the View object is used.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be

edited. If not specified, the current object is used.

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns A UI object. :

editable_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

get (**kwargs)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without notification handles

get_hashval (hash_method=None)

Return a dictionary of our items with hashes for each file.

Searches through dictionary items and if an item is a file, it calculates the md5 hash of the file contents and stores the file name and hash value as the new key value.

However, the overall bunch hash is calculated only on the hash value of a file. The path and name of the file are not used in the overall hash calculation.

Returns dict_withhash : dict

Copy of our dictionary with the new file hashes included with each file.

hashvalue : str

The md5 hash value of the traitled spec

get_traitsfree (**kwargs)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without any traits. The dictionary does not contain any attributes that were Undefined

has_metadata (name, metadata, value=None, recursive=True)

Return has_metadata for the requested trait name in this interface

has_traits_interface (*interfaces)

Returns whether the object implements a specified traits interface.

Parameters *interfaces : :

One or more traits Interface (sub)classes.

items ()

Name, trait generator for user modifiable traits

on_trait_change (handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters handler : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

on_trait_event (*handler*, *name*=None, *remove*=False, *dispatch*='same', *priority*=False, *deferred*=False, *target*=None)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters **handler** : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

print_traits (*show_help*=False, ***metadata*)

Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.

Parameters **show_help** : bool

Indicates whether to display additional descriptive information.

remove_trait (*name*)

Removes a trait attribute from this object.

Parameters **name** : str

Name of the attribute to remove.

Returns **result** : bool

True if the trait was successfully removed.

remove_trait_listener (*object*, *prefix*='')

reset_traits (*traits*=None, ***metadata*)

Resets some or all of an object's trait attributes to their default values.

Parameters **traits** : list of strings

Names of trait attributes to reset.

Returns **unresetable** : list of strings

A list of attributes that the method was unable to reset, which is empty if all the attributes were successfully reset.

set (*trait_change_notify*=True, ***traits*)

Shortcut for setting object trait attributes.

Parameters **trait_change_notify** : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: `trait_setq`)

****traits :**

Key/value pairs, the trait attributes and their values to be set

Returns self :

The method returns this object, after setting attributes.

set_trait_dispatch_handler (*name, klass, override=False*)

Sets a trait notification dispatch handler.

sync_trait (*trait_name, object, alias=None, mutual=True, remove=False*)

Synchronizes the value of a trait attribute on this object with a trait attribute on another object.

Parameters name : str

Name of the trait attribute on this object.

object : object

The object with which to synchronize.

alias : str

Name of the trait attribute on *other*; if None or omitted, same as *name*.

mutual : bool or int

Indicates whether synchronization is mutual (True or non-zero) or one-way (False or zero)

remove : bool or int

Indicates whether synchronization is being added (False or zero) or removed (True or non-zero)

trait (*name, force=False, copy=False*)

Returns the trait definition for the *name* trait attribute.

Parameters name : str

Name of the attribute whose trait definition is to be returned.

force : bool

Indicates whether to return a trait definition if *name* is not explicitly defined.

copy : bool

Indicates whether to return the original trait definition or a copy.

trait_context ()

Returns the default context to use for editing or configuring traits.

trait_get (**names, **metadata*)

Shortcut for getting object trait attributes.

Parameters names : list of strings

A list of trait attribute names whose values are requested.

Returns result : dict

A dictionary whose keys are the names passed as arguments and whose values are the corresponding trait values.

trait_items_event (*event_trait, name, items_event*)

trait_monitor (*handler, remove=False*)

Adds or removes the specified *handler* from the list of active monitors.

Parameters handler : function

The function to add or remove as a monitor.

remove : bool

Flag indicating whether to remove (True) or add the specified handler as a monitor for this class.

trait_names (***metadata*)

Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters **metadata :

Criteria for selecting trait attributes.

trait_property_changed (*name, old_value[, new_value]*)

trait_set (*trait_change_notify=True, **traits*)

Shortcut for setting object trait attributes.

Parameters **trait_change_notify** : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: `trait_setq`)

****traits** :

Key/value pairs, the trait attributes and their values to be set

Returns **self** :

The method returns this object, after setting attributes.

trait_setq (***traits*)

Shortcut for setting object trait attributes.

Parameters ****traits** :

Key/value pairs, the trait attributes and their values to be set. No trait change notifications will be generated for any values assigned (see also: `trait_set`).

Returns **self** :

The method returns this object, after setting attributes.

trait_subclasses (*all=False*)

Returns a list of the immediate (or all) subclasses of this class.

Parameters **all** : bool

Indicates whether to return all subclasses of this class. If **False**, only immediate subclasses are returned.

trait_view (*name=None, view_element=None*)

Gets or sets a ViewElement associated with an object's class.

Parameters **name** : str

Name of a view element

view_element : ViewElement

View element to associate

Returns A view element. :

trait_view_elements ()

Returns the ViewElements object associated with the object's class.

The returned object can be used to access all the view elements associated with the class.

trait_views (*klass=None*)

Returns a list of the names of all view elements associated with the current object's class.

Parameters **klass** : class

A class, such that all returned names must correspond to instances of this class. Possible values include:

Group

Item

View

ViewElement

ViewSubElement

traits (***metadata*)

Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of *metadata* criteria.

Parameters ****metadata** :

Criteria for selecting trait attributes.

traits_init ()

traits_initiated ([*True*])

validate_trait (*name, value*)

Validates whether a value is legal for a trait.

Returns the validated value if it is valid.

visible_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object, that should be GUI visible

wrappers = {'ui': <class 'traits.trait_notifiers.FastUITraitChangeNotifyWrapper'>, 'new': <class 'traits.trait_notifiers.FastUITraitChangeNotifyWrapper'>}

DynamicTraitedSpec

class `nipyre.interfaces.base.DynamicTraitedSpec` (***kwargs*)

Bases: `nipyre.interfaces.base.BaseTraitedSpec`

A subclass to handle dynamic traits

This class is a workaround for `add_traits` and `clone_traits` not functioning well together.

Methods

<code>add_class_trait(name, *trait)</code>	Adds a named trait attribute to this class.
<code>add_trait(name, *trait)</code>	Adds a trait attribute to this object.
<code>add_trait_category(category)</code>	Adds a trait category to a class.
<code>add_trait_listener(object[, prefix])</code>	
<code>all_trait_names()</code>	Returns the list of all trait names, including implicitly defined traits.
<code>base_trait(name)</code>	Returns the base trait definition for a trait attribute.
<code>class_default_traits_view()</code>	Returns the name of the default traits view for the class.
<code>class_editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<code>class_trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>class_trait_view([name, view_element])</code>	
<code>class_trait_view_elements()</code>	Returns the ViewElements object associated with the class.
<code>class_traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
<code>class_visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event
<code>clone_traits([traits, memo, copy])</code>	Clones a new object from this one, optionally copying only a specified set of traits.
<code>configure_traits([filename, view, kind, ...])</code>	Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
<code>copy_traits(other[, traits, memo, copy])</code>	Copies another object's trait attributes into this one.
<code>copyable_trait_names(**metadata)</code>	Returns the list of trait names to copy or clone by default.
<code>default_traits_view()</code>	Returns the name of the default traits view for the object's class.
<code>edit_traits([view, parent, kind, context, ...])</code>	Displays a user interface window for editing trait attribute values.
<code>editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.
<code>get(**kwargs)</code>	Returns traited class as a dict
<code>get_hashval([hash_method])</code>	Return a dictionary of our items with hashes for each file.
<code>get_traitsfree(**kwargs)</code>	Returns traited class as a dict

Continued on next page

Table 3.11 – continued from previous page

<code>has_metadata(name, metadata[, value, recursive])</code>	Return <code>has_metadata</code> for the requested trait name in this
<code>has_traits_interface(*interfaces)</code>	Returns whether the object implements a specified traits interface.
<code>items()</code>	Name, trait generator for user modifiable traits
<code>on_trait_change(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>on_trait_event(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>print_traits([show_help])</code>	Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.
<code>remove_trait(name)</code>	Removes a trait attribute from this object.
<code>remove_trait_listener(object[, prefix])</code>	
<code>reset_traits([traits])</code>	Resets some or all of an object's trait attributes to their default values.
<code>set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>set_trait_dispatch_handler(name, klass[, ...])</code>	Sets a trait notification dispatch handler.
<code>sync_trait(trait_name, object[, alias, ...])</code>	Synchronizes the value of a trait attribute on this object with a trait attribute on another object.
<code>trait(name[, force, copy])</code>	Returns the trait definition for the <i>name</i> trait attribute.
<code>trait_context()</code>	Returns the default context to use for editing or configuring traits.
<code>trait_get(*names, **metadata)</code>	Shortcut for getting object trait attributes.
<code>trait_items_event(event_trait, name, items_event)</code>	
<code>trait_monitor(handler[, remove])</code>	Adds or removes the specified <i>handler</i> from the list of active monitors.
<code>trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>trait_property_changed(...)</code>	
<code>trait_set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>trait_setq(**traits)</code>	Shortcut for setting object trait attributes.
<code>trait_subclasses([all])</code>	Returns a list of the immediate (or all) subclasses of this class.
<code>trait_view([name, view_element])</code>	Gets or sets a <code>ViewElement</code> associated with an object's class.
<code>trait_view_elements()</code>	Returns the <code>ViewElements</code> object associated with the object's class.
<code>trait_views([klass])</code>	Returns a list of the names of all view elements associated with the current object's class.
<code>traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of <i>metadata</i> criteria.
<code>traits_init()</code>	
<code>traits_inited([True])</code>	
<code>validate_trait(name, value)</code>	Validates whether a value is legal for a trait.

Continued on next page

Table 3.11 – continued from previous page

<code>visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event
<hr/>	
<code>__init__</code> (<i>**kwargs</i>)	Initialize handlers and inputs
<code>add_class_trait</code> (<i>name</i> , <i>*trait</i>)	Adds a named trait attribute to this class.
Parameters <i>name</i> : str	Name of the attribute to add.
<i>*trait</i> :	A trait or a value that can be converted to a trait using <code>Trait()</code> . Trait definition of the attribute. It can be a single value or a list equivalent to an argument list for the <code>Trait()</code> function.
<code>add_trait</code> (<i>name</i> , <i>*trait</i>)	Adds a trait attribute to this object.
Parameters <i>name</i> : str	Name of the attribute to add.
<i>*trait</i> :	Trait or a value that can be converted to a trait by <code>Trait()</code> . Trait definition for <i>name</i> . If more than one value is specified, it is equivalent to passing the entire list of values to <code>Trait()</code> .
<code>add_trait_category</code> (<i>category</i>)	Adds a trait category to a class.
<code>add_trait_listener</code> (<i>object</i> , <i>prefix</i> = <i>''</i>)	
<code>all_trait_names</code> ()	Returns the list of all trait names, including implicitly defined traits.
<code>base_trait</code> (<i>name</i>)	Returns the base trait definition for a trait attribute.
Parameters <i>name</i> : str	Name of the attribute whose trait definition is returned.
<code>class_default_traits_view</code> ()	Returns the name of the default traits view for the class.
<code>class_editable_traits</code> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<code>class_trait_names</code> (<i>**metadata</i>)	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
Parameters <i>**metadata</i> :	Criteria for selecting trait attributes.
<code>class_trait_view</code> (<i>name</i> = <i>None</i> , <i>view_element</i> = <i>None</i>)	
<code>class_trait_view_elements</code> ()	Returns the <code>ViewElements</code> object associated with the class.
	The returned object can be used to access all the view elements associated with the class.
<code>class_traits</code> (<i>**metadata</i>)	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
Parameters <i>**metadata</i> :	Criteria for selecting trait attributes.
<code>class_visible_traits</code> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class, that should be GUI visible
<code>clone_traits</code> (<i>traits</i> = <i>None</i> , <i>memo</i> = <i>None</i> , <i>copy</i> = <i>None</i> , <i>**metadata</i>)	

Clones a new object from this one, optionally copying only a specified set of traits.

Parameters **traits** : list of strings

The list of names of the trait attributes to copy.

memo : dict

A dictionary of objects that have already been copied.

copy : str

The type of copy `deep` or `shallow` to perform on any trait that does not have explicit 'copy' metadata. A value of `None` means 'copy reference'.

Returns **new** :

The newly cloned object.

configure_traits (*filename=None, view=None, kind=None, edit=True, context=None, handler=None, id='', scrollable=None, **args*)

Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.

Parameters **filename** : str

The name (including path) of a file that contains a pickled representation of the current object. When this parameter is specified, the method reads the corresponding file (if it exists) to restore the saved values of the object's traits before displaying them. If the user confirms the dialog box (by clicking **OK**), the new values are written to the file. If this parameter is not specified, the values are loaded from the in-memory object, and are not persisted when the dialog box is closed.

view : View or str

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

kind : str

The type of user interface window to create. See the **traitsui.view.kind_trait** trait for values and their meanings. If *kind* is unspecified or `None`, the **kind** attribute of the View object is used.

edit : bool

Indicates whether to display a user interface. If *filename* specifies an existing file, setting *edit* to `False` loads the saved values from that file into the object without requiring user interaction.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used

handler : Handler

A handler object used for event handling in the dialog box. If `None`, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to `True`, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns **True on success.** :

copy_traits (*other, traits=None, memo=None, copy=None, **metadata*)

Copies another object's trait attributes into this one.

Parameters **other** : object

The object whose trait attribute values should be copied.

traits : list of strings

A list of names of trait attributes to copy. If `None` or unspecified, the set of names

returned by `trait_names()` is used. If 'all' or an empty list, the set of names returned by `all_trait_names()` is used.

memo : dict

A dictionary of objects that have already been copied.

copy : None | 'deep' | 'shallow'

The type of copy to perform on any trait that does not have explicit 'copy' metadata.

A value of None means 'copy reference'.

Returns unassignable : list of strings

A list of attributes that the method was unable to copy, which is empty if all the attributes were successfully copied.

copyable_trait_names (***metadata*)

Returns the list of trait names to copy or clone by default.

default_traits_view ()

Returns the name of the default traits view for the object's class.

edit_traits (*view=None, parent=None, kind=None, context=None, handler=None, id='', scrollable=None, **args*)

Displays a user interface window for editing trait attribute values.

Parameters view : View or string

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

parent : toolkit control

The reference to a user interface component to use as the parent window for the object's UI window.

kind : str

The type of user interface window to create. See the **traitsui.view.kind_trait** trait for values and their meanings. If *kind* is unspecified or None, the **kind** attribute of the View object is used.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used.

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns A UI object. :

editable_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

get (***kwargs*)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without notification handles

get_hashval (*hash_method=None*)

Return a dictionary of our items with hashes for each file.

Searches through dictionary items and if an item is a file, it calculates the md5 hash of the file contents and stores the file name and hash value as the new key value.

However, the overall bunch hash is calculated only on the hash value of a file. The path and name of

the file are not used in the overall hash calculation.

Returns **dict_withhash** : dict

Copy of our dictionary with the new file hashes included with each file.

hashvalue : str

The md5 hash value of the traited spec

get_traitsfree (***kwargs*)

Returns traited class as a dict

Augments the trait get function to return a dictionary without any traits. The dictionary does not contain any attributes that were Undefined

has_metadata (*name, metadata, value=None, recursive=True*)

Return has_metadata for the requested trait name in this interface

has_traits_interface (**interfaces*)

Returns whether the object implements a specified traits interface.

Parameters ***interfaces** :

One or more traits Interface (sub)classes.

items ()

Name, trait generator for user modifiable traits

on_trait_change (*handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters **handler** : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

on_trait_event (*handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters **handler** : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

print_traits (*show_help=False, **metadata*)

Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.

Parameters *show_help* : bool

Indicates whether to display additional descriptive information.

remove_trait (*name*)

Removes a trait attribute from this object.

Parameters *name* : str

Name of the attribute to remove.

Returns *result* : bool

True if the trait was successfully removed.

remove_trait_listener (*object, prefix=''*)

reset_traits (*traits=None, **metadata*)

Resets some or all of an object's trait attributes to their default values.

Parameters *traits* : list of strings

Names of trait attributes to reset.

Returns *unresetable* : list of strings

A list of attributes that the method was unable to reset, which is empty if all the attributes were successfully reset.

set (*trait_change_notify=True, **traits*)

Shortcut for setting object trait attributes.

Parameters *trait_change_notify* : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: *trait_setq*)

****traits** :

Key/value pairs, the trait attributes and their values to be set

Returns *self* :

The method returns this object, after setting attributes.

set_trait_dispatch_handler (*name, klass, override=False*)

Sets a trait notification dispatch handler.

sync_trait (*trait_name, object, alias=None, mutual=True, remove=False*)

Synchronizes the value of a trait attribute on this object with a trait attribute on another object.

Parameters *name* : str

Name of the trait attribute on this object.

object : object

The object with which to synchronize.

alias : str

Name of the trait attribute on *other*; if None or omitted, same as *name*.

mutual : bool or int

Indicates whether synchronization is mutual (True or non-zero) or one-way (False or zero)

remove : bool or int

Indicates whether synchronization is being added (False or zero) or removed (True or non-zero)

trait (*name, force=False, copy=False*)

Returns the trait definition for the *name* trait attribute.

Parameters *name* : str

Name of the attribute whose trait definition is to be returned.

force : bool

Indicates whether to return a trait definition if *name* is not explicitly defined.

copy : bool

Indicates whether to return the original trait definition or a copy.

trait_context ()

Returns the default context to use for editing or configuring traits.

trait_get (**names*, ***metadata*)

Shortcut for getting object trait attributes.

Parameters *names* : list of strings

A list of trait attribute names whose values are requested.

Returns *result* : dict

A dictionary whose keys are the names passed as arguments and whose values are the corresponding trait values.

trait_items_event (*event_trait*, *name*, *items_event*)

trait_monitor (*handler*, *remove=False*)

Adds or removes the specified *handler* from the list of active monitors.

Parameters *handler* : function

The function to add or remove as a monitor.

remove : bool

Flag indicating whether to remove (True) or add the specified handler as a monitor for this class.

trait_names (***metadata*)

Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters ***metadata* :

Criteria for selecting trait attributes.

trait_property_changed (*name*, *old_value* [, *new_value*])

trait_set (*trait_change_notify=True*, ***traits*)

Shortcut for setting object trait attributes.

Parameters *trait_change_notify* : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: *trait_setq*)

****traits** :

Key/value pairs, the trait attributes and their values to be set

Returns *self* :

The method returns this object, after setting attributes.

trait_setq (***traits*)

Shortcut for setting object trait attributes.

Parameters ***traits* :

Key/value pairs, the trait attributes and their values to be set. No trait change notifications will be generated for any values assigned (see also: *trait_set*).

Returns *self* :

The method returns this object, after setting attributes.

trait_subclasses (*all=False*)

Returns a list of the immediate (or all) subclasses of this class.

Parameters *all* : bool

Indicates whether to return all subclasses of this class. If **False**, only immediate subclasses are returned.

trait_view (*name=None*, *view_element=None*)

Gets or sets a *ViewElement* associated with an object's class.

Parameters *name* : str

Name of a view element
view_element : ViewElement
 View element to associate
Returns A view element. :
trait_view_elements ()
 Returns the ViewElements object associated with the object's class.
 The returned object can be used to access all the view elements associated with the class.
trait_views (klass=None)
 Returns a list of the names of all view elements associated with the current object's class.
Parameters klass : class
 A class, such that all returned names must correspond to instances of this class. Possible values include:
 Group
 Item
 View
 ViewElement
 ViewSubElement
traits (**metadata)
 Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of *metadata* criteria.
Parameters **metadata : :
 Criteria for selecting trait attributes.
traits_init ()
traits_initiated ([True])
validate_trait (name, value)
 Validates whether a value is legal for a trait.
 Returns the validated value if it is valid.
visible_traits ()
 Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object, that should be GUI visible
wrappers = {'ui': <class 'traits.trait_notifiers.FastUITraitChangeNotifyWrapper'>, 'new': <class 'traits.trait_notifiers.NewWrapper'>}

InputMultiPath

```
class nipy.interfaces.base.InputMultiPath (trait=None, value=None, minlen=0,
                                             maxlen=9223372036854775807,
                                             items=True, **metadata)
```

Bases: `nipy.interfaces.base.MultiPath`

Implements a user friendly traits that accepts one or more paths to files or directories. This is the input version which always returns a list. Default value of this trait is `_Undefined`. It does not accept empty lists. XXX This should only be used as a final resort. We should stick to established Traits to the extent possible. XXX This needs to be vetted by somebody who understands traits

```
>>> from nipy.interfaces.base import InputMultiPath
>>> class A(TraitSpec):
...     foo = InputMultiPath(File(exists=False))
>>> a = A()
>>> a.foo
<undefined>
```

```
>>> a.foo = '/software/temp/foo.txt'
>>> a.foo
['/software/temp/foo.txt']
```

```
>>> a.foo = ['/software/temp/foo.txt']
>>> a.foo
['/software/temp/foo.txt']
```

```
>>> a.foo = ['/software/temp/foo.txt', '/software/temp/goo.txt']
>>> a.foo
['/software/temp/foo.txt', '/software/temp/goo.txt']
```

Attributes

editor

info_trait

Methods

<code>__call__(*args, **kw)</code>	Allows a derivative trait to be defined from this one.
<code>as_ctrait()</code>	Returns a CTrait corresponding to the trait defined by this class.
<code>clone([default_value])</code>	Clones the contents of this object into a new instance of the same class, and then modifies the cloned copy using the specified <i>default_value</i> and <i>metadata</i> .
<code>create_editor()</code>	Returns the default UI editor for the trait.
<code>error(object, name, value)</code>	Raises a TraitError exception.
<code>full_info(object, name, value)</code>	Returns a description of the trait.
<code>get_default_value()</code>	Returns a tuple of the form: (<i>default_value_type</i> , <i>default_value</i>) which describes the default value for this trait.
<code>get_editor([trait])</code>	Returns a trait editor that allows the user to modify the <i>trait</i> trait.
<code>get_value(object, name[, trait])</code>	Returns the current value of a property-based trait.
<code>info()</code>	Must return a string describing the type of value accepted by the trait handler.
<code>init()</code>	Allows the trait to perform any additional initialization needed.
<code>inner_traits()</code>	Returns the <i>inner trait</i> (or traits) for this trait.
<code>is_valid(object, name, value)</code>	
<code>items_event()</code>	
<code>repr(value)</code>	Returns a printable representation of a value along with its type.
<code>set_value(object, name, value)</code>	Sets the cached value of a property-based trait and fires the appropriate trait change event.
<code>validate(object, name, value)</code>	

`__init__` (*trait=None*, *value=None*, *minlen=0*, *maxlen=9223372036854775807*, *items=True*, ***metadata*)
Returns a List trait.

Parameters **trait** : a trait or value that can be converted to a trait using Trait()

The type of item that the list contains. If not specified, the list can contain items of any type.

value : list

Default value for the list.
minlen : integer
The minimum length of a list that can be assigned to the trait.
maxlen : integer
The maximum length of a list that can be assigned to the trait.
The length of the list assigned to the trait must be such that:: :
minlen <= len(list) <= maxlen

as_ctrait ()

Returns a CTrait corresponding to the trait defined by this class.

clone (*default_value*=<missing>, ***metadata*)

Clones the contents of this object into a new instance of the same class, and then modifies the cloned copy using the specified *default_value* and *metadata*. Returns the cloned object as the result.

Note that subclasses can change the signature of this method if needed, but should always call the ‘super’ method if possible.

create_editor ()

Returns the default UI editor for the trait.

default_value = <undefined>

default_value_type = 5

editor = None

error (*object*, *name*, *value*)

Raises a TraitError exception.

Parameters **object** : object

The object whose attribute is being assigned.

name : str

The name of the attribute being assigned.

value : object

The proposed new value for the attribute.

full_info (*object*, *name*, *value*)

Returns a description of the trait.

get_default_value ()

Returns a tuple of the form: (*default_value_type*, *default_value*) which describes the default value for this trait. The default implementation analyzes the value of the trait’s **default_value** attribute and determines an appropriate *default_value_type* for *default_value*. If you need to override this method to provide a different result tuple, the following values are valid values for *default_value_type*:

0, 1: The *default_value* item of the tuple is the default value.

2: The object containing the trait is the default value.

3: A new copy of the list specified by *default_value* is the default value.

4: A new copy of the dictionary specified by *default_value* is the default value.

5: A new instance of TraitListObject constructed using the *default_value* list is the default value.

6: A new instance of TraitDictObject constructed using the *default_value* dictionary is the default value.

7: *default_value* is a tuple of the form: (*callable*, *args*, *kw*), where *callable* is a callable, *args* is a tuple, and *kw* is either a dictionary or None. The default value is the result obtained by invoking *callable*(**args*, ***kw*).

8: *default_value* is a callable. The default value is the result obtained by invoking *default_value*(**object*), where *object* is the object containing the trait. If the trait has a *validate*()

method, the `validate()` method is also called to validate the result.

9: A new instance of `TraitSetObject` constructed using the `default_value` set is the default value.

get_editor (*trait=None*)

Returns a trait editor that allows the user to modify the *trait* trait.

Parameters *trait* : Trait

The trait to be edited.

get_value (*object, name, trait=None*)

Returns the current value of a property-based trait.

has_items = False

info ()

Must return a string describing the type of value accepted by the trait handler.

The string should be a phrase describing the type defined by the `TraitHandler` subclass, rather than a complete sentence. For example, use the phrase, “a square sprocket” instead of the sentence, “The value must be a square sprocket.” The value returned by `info()` is combined with other information whenever an error occurs and therefore makes more sense to the user if the result is a phrase. The `info()` method is similar in purpose and use to the **info** attribute of a validator function.

Note that the result can include information specific to the particular trait handler instance. For example, `TraitRange` instances return a string indicating the range of values acceptable to the handler (e.g., “an integer in the range from 1 to 9”). If the `info()` method is not overridden, the default method returns the value of the ‘`info_text`’ attribute.

info_text = ‘a legal value’

info_trait = None

init ()

Allows the trait to perform any additional initialization needed.

inner_traits ()

Returns the *inner trait* (or traits) for this trait.

is_mapped = False

is_valid (*object, name, value*)

items_event ()

metadata = {}

repr (*value*)

Returns a printable representation of a value along with its type.

Deprecated since version 3.0.3: This functionality was only used to provide readable error messages.

This functionality has been incorporated into `TraitError` itself.

Parameters *value* : object

The value to be printed.

set_value (*object, name, value*)

Sets the cached value of a property-based trait and fires the appropriate trait change event.

validate (*object, name, value*)

Interface

class `nipytype.interfaces.base.Interface` (***inputs*)

Bases: `object`

This is an abstract definition for Interface objects.

It provides no functionality. It defines the necessary attributes and methods all Interface objects should have.

Attributes

input_spec

Continued on next page

Table 3.14 – continued from previous page

<i>output_spec</i>	
Methods	
<i>aggregate_outputs</i> ([runtime, needed_outputs])	Called to populate outputs
<i>help</i> ()	Prints class help
<i>run</i> ()	Execute the command.

```

__init__ (**inputs)
    Initialize command with given args and inputs.
aggregate_outputs (runtime=None, needed_outputs=None)
    Called to populate outputs
always_run
can_resume
classmethod help ()
    Prints class help
input_spec = None
output_spec = None
run ()
    Execute the command.
version

```

InterfaceResult

class nipyne.interfaces.base.**InterfaceResult** (*interface, runtime, inputs=None, outputs=None, provenance=None*)

Bases: object

Object that contains the results of running a particular Interface.

Attributes**MpiCommandLine**

class nipyne.interfaces.base.**MpiCommandLine** (*command=None, **inputs*)

Bases: *nipyne.interfaces.base.CommandLine*

Implements functionality to interact with command line programs that can be run with MPI (i.e. using 'mpiexec').

Examples

```

>>> from nipyne.interfaces.base import MpiCommandLine
>>> mpi_cli = MpiCommandLine(command='my_mpi_prog')
>>> mpi_cli.inputs.args = '-v'
>>> mpi_cli.cmdline
'my_mpi_prog -v'

```

```

>>> mpi_cli.inputs.use_mpi = True
>>> mpi_cli.inputs.n_procs = 8

```

```
>>> mpi_cli.cmdline
'mpiexec -n 8 my_mpi_prog -v'
```

Attributes

output_spec

Methods

<i>aggregate_outputs</i> (<i>runtime</i> , <i>needed_outputs</i>)	Collate expected outputs and check for existence
<i>help</i> (<i>returnhelp</i>)	
<i>input_spec</i>	alias of <i>MpiCommandLineInputSpec</i>
<i>load_inputs_from_json</i> (<i>json_file</i> [, <i>overwrite</i>])	A convenient way to load pre-set inputs from a JSON file.
<i>raise_exception</i> (<i>runtime</i>)	
<i>run</i> (** <i>inputs</i>)	Execute this interface.
<i>save_inputs_to_json</i> (<i>json_file</i>)	A convenient way to save current inputs to a JSON file.
<i>set_default_terminal_output</i> (<i>output_type</i>)	Set the default terminal output for CommandLine Interfaces.
<i>version_from_command</i> (<i>flag</i>)	

```
__init__(command=None, **inputs)
aggregate_outputs (runtime=None, needed_outputs=None)
    Collate expected outputs and check for existence
always_run
can_resume
cmd
    sets base command, immutable
cmdline
    Adds 'mpiexec' to beginning of command
help (returnhelp=False)
input_spec
    alias of MpiCommandLineInputSpec
load_inputs_from_json (json_file, overwrite=True)
    A convenient way to load pre-set inputs from a JSON file.
output_spec = None
raise_exception (runtime)
references_ = []
run (**inputs)
    Execute this interface.
    This interface will not raise an exception if runtime.returncode is non-zero.
    Parameters inputs : allows the interface settings to be updated
    Returns results : an InterfaceResult object containing a copy of the instance
    that was executed, provenance information and, if successful, results :
save_inputs_to_json (json_file)
    A convenient way to save current inputs to a JSON file.
set_default_terminal_output (output_type)
    Set the default terminal output for CommandLine Interfaces.
```

This method is used to set default terminal output for CommandLine Interfaces. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.terminal_output.

```
version
version_from_command(flag='-v')
```

MpiCommandLineInputSpec

```
class nipyre.interfaces.base.MpiCommandLineInputSpec(**kwargs)
```

Bases: *nipyre.interfaces.base.CommandLineInputSpec*

Methods

<i>add_class_trait</i> (name, *trait)	Adds a named trait attribute to this class.
<i>add_trait</i> (name, *trait)	Adds a trait attribute to this object.
<i>add_trait_category</i> (category)	Adds a trait category to a class.
<i>add_trait_listener</i> (object[, prefix])	
<i>all_trait_names</i> ()	Returns the list of all trait names, including implicitly defined traits.
<i>base_trait</i> (name)	Returns the base trait definition for a trait attribute.
<i>class_default_traits_view</i> ()	Returns the name of the default traits view for the class.
<i>class_editable_traits</i> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<i>class_trait_names</i> (**metadata)	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<i>class_trait_view</i> ([name, view_element])	
<i>class_trait_view_elements</i> ()	Returns the ViewElements object associated with the class.
<i>class_traits</i> (**metadata)	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
<i>class_visible_traits</i> ()	Returns an alphabetically sorted list of the names of non-event
<i>clone_traits</i> ([traits, memo, copy])	Clones a new object from this one, optionally copying only a specified set of traits.
<i>configure_traits</i> ([filename, view, kind, ...])	Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
<i>copy_traits</i> (other[, traits, memo, copy])	Copies another object's trait attributes into this one.
<i>copyable_trait_names</i> (**metadata)	Returns the list of trait names to copy or clone by default.
<i>default_traits_view</i> ()	Returns the name of the default traits view for the object's class.
<i>edit_traits</i> ([view, parent, kind, context, ...])	Displays a user interface window for editing trait attribute values.
<i>editable_traits</i> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

Continued on next page

Table 3.19 – continued from previous page

<code>get(**kwargs)</code>	Returns traitled class as a dict
<code>get_hashval([hash_method])</code>	Return a dictionary of our items with hashes for each file.
<code>get_traitsfree(**kwargs)</code>	Returns traitled class as a dict
<code>has_metadata(name, metadata[, value, recursive])</code>	Return <code>has_metadata</code> for the requested trait name in this
<code>has_traits_interface(*interfaces)</code>	Returns whether the object implements a specified traits interface.
<code>items()</code>	Name, trait generator for user modifiable traits
<code>on_trait_change(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>on_trait_event(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>print_traits([show_help])</code>	Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.
<code>remove_trait(name)</code>	Removes a trait attribute from this object.
<code>remove_trait_listener(object[, prefix])</code>	
<code>reset_traits([traits])</code>	Resets some or all of an object's trait attributes to their default values.
<code>set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>set_trait_dispatch_handler(name, klass[, ...])</code>	Sets a trait notification dispatch handler.
<code>sync_trait(trait_name, object[, alias, ...])</code>	Synchronizes the value of a trait attribute on this object with a trait attribute on another object.
<code>trait(name[, force, copy])</code>	Returns the trait definition for the <i>name</i> trait attribute.
<code>trait_context()</code>	Returns the default context to use for editing or configuring traits.
<code>trait_get(*names, **metadata)</code>	Shortcut for getting object trait attributes.
<code>trait_items_event(event_trait, name, items_event)</code>	
<code>trait_monitor(handler[, remove])</code>	Adds or removes the specified <i>handler</i> from the list of active monitors.
<code>trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>trait_property_changed(...)</code>	
<code>trait_set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>trait_setq(**traits)</code>	Shortcut for setting object trait attributes.
<code>trait_subclasses([all])</code>	Returns a list of the immediate (or all) subclasses of this class.
<code>trait_view([name, view_element])</code>	Gets or sets a <code>ViewElement</code> associated with an object's class.
<code>trait_view_elements()</code>	Returns the <code>ViewElements</code> object associated with the object's class.
<code>trait_views([klass])</code>	Returns a list of the names of all view elements associated with the current object's class.
<code>traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of <i>metadata</i> criteria.

Continued on next page

Table 3.19 – continued from previous page

<code>traits_init()</code>	
<code>traits_inited([True])</code>	
<code>validate_trait(name, value)</code>	Validates whether a value is legal for a trait.
<code>visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event

`__init__` (**kwargs)

Initialize handlers and inputs

`add_class_trait` (name, *trait)

Adds a named trait attribute to this class.

Parameters name : str

Name of the attribute to add.

*trait :

A trait or a value that can be converted to a trait using Trait() Trait definition of the attribute. It can be a single value or a list equivalent to an argument list for the Trait() function.

`add_trait` (name, *trait)

Adds a trait attribute to this object.

Parameters name : str

Name of the attribute to add.

*trait :

Trait or a value that can be converted to a trait by Trait(). Trait definition for name. If more than one value is specified, it is equivalent to passing the entire list of values to Trait().

`add_trait_category` (category)

Adds a trait category to a class.

`add_trait_listener` (object, prefix='')

`all_trait_names` ()

Returns the list of all trait names, including implicitly defined traits.

`base_trait` (name)

Returns the base trait definition for a trait attribute.

Parameters name : str

Name of the attribute whose trait definition is returned.

`class_default_traits_view` ()

Returns the name of the default traits view for the class.

`class_editable_traits` ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.

`class_trait_names` (**metadata)

Returns a list of the names of all trait attributes whose definitions match the set of metadata criteria specified.

Parameters **metadata :

Criteria for selecting trait attributes.

`class_trait_view` (name=None, view_element=None)

`class_trait_view_elements` ()

Returns the ViewElements object associated with the class.

The returned object can be used to access all the view elements associated with the class.

`class_traits` (**metadata)

Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of metadata criteria.

Parameters **metadata :

Criteria for selecting trait attributes.

`class_visible_traits` ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class, that should be GUI visible

clone_traits (*traits=None, memo=None, copy=None, **metadata*)

Clones a new object from this one, optionally copying only a specified set of traits.

Parameters **traits** : list of strings

The list of names of the trait attributes to copy.

memo : dict

A dictionary of objects that have already been copied.

copy : str

The type of copy `deep` or `shallow` to perform on any trait that does not have explicit 'copy' metadata. A value of `None` means 'copy reference'.

Returns **new** :

The newly cloned object.

configure_traits (*filename=None, view=None, kind=None, edit=True, context=None, handler=None, id='', scrollable=None, **args*)

Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.

Parameters **filename** : str

The name (including path) of a file that contains a pickled representation of the current object. When this parameter is specified, the method reads the corresponding file (if it exists) to restore the saved values of the object's traits before displaying them. If the user confirms the dialog box (by clicking **OK**), the new values are written to the file. If this parameter is not specified, the values are loaded from the in-memory object, and are not persisted when the dialog box is closed.

view : View or str

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or `None`, the **kind** attribute of the View object is used.

edit : bool

Indicates whether to display a user interface. If *filename* specifies an existing file, setting *edit* to `False` loads the saved values from that file into the object without requiring user interaction.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used

handler : Handler

A handler object used for event handling in the dialog box. If `None`, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to `True`, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns **True on success.** :

copy_traits (*other, traits=None, memo=None, copy=None, **metadata*)

Copies another object's trait attributes into this one.

Parameters **other** : object

The object whose trait attribute values should be copied.

traits : list of strings

A list of names of trait attributes to copy. If None or unspecified, the set of names returned by `trait_names()` is used. If 'all' or an empty list, the set of names returned by `all_trait_names()` is used.

memo : dict

A dictionary of objects that have already been copied.

copy : None | 'deep' | 'shallow'

The type of copy to perform on any trait that does not have explicit 'copy' metadata. A value of None means 'copy reference'.

Returns unassignable : list of strings

A list of attributes that the method was unable to copy, which is empty if all the attributes were successfully copied.

copyable_trait_names (***metadata*)

Returns the list of trait names to copy or clone by default.

default_traits_view ()

Returns the name of the default traits view for the object's class.

edit_traits (*view=None, parent=None, kind=None, context=None, handler=None, id='', scrollable=None, **args*)

Displays a user interface window for editing trait attribute values.

Parameters view : View or string

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

parent : toolkit control

The reference to a user interface component to use as the parent window for the object's UI window.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or None, the **kind** attribute of the View object is used.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used.

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns A UI object. :

editable_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

get (***kwargs*)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without notification handles

get_hashval (*hash_method=None*)

Return a dictionary of our items with hashes for each file.

Searches through dictionary items and if an item is a file, it calculates the md5 hash of the file contents and stores the file name and hash value as the new key value.

However, the overall bunch hash is calculated only on the hash value of a file. The path and name of the file are not used in the overall hash calculation.

Returns dict_withhash : dict

Copy of our dictionary with the new file hashes included with each file.

hashvalue : str

The md5 hash value of the traited spec

get_traitsfree (**kwargs)

Returns traited class as a dict

Augments the trait get function to return a dictionary without any traits. The dictionary does not contain any attributes that were Undefined

has_metadata (name, metadata, value=None, recursive=True)

Return has_metadata for the requested trait name in this interface

has_traits_interface (*interfaces)

Returns whether the object implements a specified traits interface.

Parameters *interfaces :

One or more traits Interface (sub)classes.

items ()

Name, trait generator for user modifiable traits

on_trait_change (handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters handler : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

on_trait_event (handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters handler : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

print_traits (*show_help=False, **metadata*)

Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.

Parameters *show_help* : bool

Indicates whether to display additional descriptive information.

remove_trait (*name*)

Removes a trait attribute from this object.

Parameters *name* : str

Name of the attribute to remove.

Returns *result* : bool

True if the trait was successfully removed.

remove_trait_listener (*object, prefix=''*)

reset_traits (*traits=None, **metadata*)

Resets some or all of an object's trait attributes to their default values.

Parameters *traits* : list of strings

Names of trait attributes to reset.

Returns *unresetable* : list of strings

A list of attributes that the method was unable to reset, which is empty if all the attributes were successfully reset.

set (*trait_change_notify=True, **traits*)

Shortcut for setting object trait attributes.

Parameters *trait_change_notify* : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: *trait_setq*)

****traits** :

Key/value pairs, the trait attributes and their values to be set

Returns *self* :

The method returns this object, after setting attributes.

set_trait_dispatch_handler (*name, klass, override=False*)

Sets a trait notification dispatch handler.

sync_trait (*trait_name, object, alias=None, mutual=True, remove=False*)

Synchronizes the value of a trait attribute on this object with a trait attribute on another object.

Parameters *name* : str

Name of the trait attribute on this object.

object : object

The object with which to synchronize.

alias : str

Name of the trait attribute on *other*; if None or omitted, same as *name*.

mutual : bool or int

Indicates whether synchronization is mutual (True or non-zero) or one-way (False or zero)

remove : bool or int

Indicates whether synchronization is being added (False or zero) or removed (True or non-zero)

trait (*name*, *force=False*, *copy=False*)

Returns the trait definition for the *name* trait attribute.

Parameters **name** : str

Name of the attribute whose trait definition is to be returned.

force : bool

Indicates whether to return a trait definition if *name* is not explicitly defined.

copy : bool

Indicates whether to return the original trait definition or a copy.

trait_context ()

Returns the default context to use for editing or configuring traits.

trait_get (**names*, ***metadata*)

Shortcut for getting object trait attributes.

Parameters **names** : list of strings

A list of trait attribute names whose values are requested.

Returns **result** : dict

A dictionary whose keys are the names passed as arguments and whose values are the corresponding trait values.

trait_items_event (*event_trait*, *name*, *items_event*)

trait_monitor (*handler*, *remove=False*)

Adds or removes the specified *handler* from the list of active monitors.

Parameters **handler** : function

The function to add or remove as a monitor.

remove : bool

Flag indicating whether to remove (True) or add the specified handler as a monitor for this class.

trait_names (***metadata*)

Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters ****metadata** :

Criteria for selecting trait attributes.

trait_property_changed (*name*, *old_value* [, *new_value*])

trait_set (*trait_change_notify=True*, ***traits*)

Shortcut for setting object trait attributes.

Parameters **trait_change_notify** : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: *trait_setq*)

****traits** :

Key/value pairs, the trait attributes and their values to be set

Returns **self** :

The method returns this object, after setting attributes.

trait_setq (***traits*)

Shortcut for setting object trait attributes.

Parameters ****traits** :

Key/value pairs, the trait attributes and their values to be set. No trait change notifications will be generated for any values assigned (see also: *trait_set*).

Returns **self** :

The method returns this object, after setting attributes.

trait_subclasses (*all=False*)

Returns a list of the immediate (or all) subclasses of this class.

Parameters **all** : bool

Indicates whether to return all subclasses of this class. If False, only immediate subclasses are returned.

```

trait_view (name=None, view_element=None)
    Gets or sets a ViewElement associated with an object’s class.
    Parameters name : str
        Name of a view element
    view_element : ViewElement
        View element to associate
    Returns A view element. :
trait_view_elements ()
    Returns the ViewElements object associated with the object’s class.
    The returned object can be used to access all the view elements associated with the class.
trait_views (klass=None)
    Returns a list of the names of all view elements associated with the current object’s class.
    Parameters klass : class
        A class, such that all returned names must correspond to instances of this class. Pos-
        sible values include:
        Group
        Item
        View
        ViewElement
        ViewSubElement
traits (**metadata)
    Returns a dictionary containing the definitions of all of the trait attributes of this object that match the
    set of metadata criteria.
    Parameters **metadata :
        Criteria for selecting trait attributes.
traits_init ()
traits_initiated ([True ])
validate_trait (name, value)
    Validates whether a value is legal for a trait.
    Returns the validated value if it is valid.
visible_traits ()
    Returns an alphabetically sorted list of the names of non-event trait attributes associated with the
    current object, that should be GUI visible
wrappers = {'ui': <class ‘traits.trait_notifiers.FastUITraitChangeNotifyWrapper’>, ‘new’: <class ‘traits.trait_notifi

```

MultiPath

```

class nipy.interfaces.base.MultiPath (trait=None, value=None, minlen=0,
                                       maxlen=9223372036854775807, items=True,
                                       **metadata)

```

Bases: traits.trait_types.List

Abstract class - shared functionality of input and output MultiPath

Attributes

editor

info_trait

Methods

<code>__call__(*args, **kw)</code>	Allows a derivative trait to be defined from this one.
------------------------------------	--

Continued on next page

Table 3.21 – continued from previous page

<code>as_ctrait()</code>	Returns a CTrait corresponding to the trait defined by this class.
<code>clone([default_value])</code>	Clones the contents of this object into a new instance of the same class, and then modifies the cloned copy using the specified <i>default_value</i> and <i>metadata</i> .
<code>create_editor()</code>	Returns the default UI editor for the trait.
<code>error(object, name, value)</code>	Raises a TraitError exception.
<code>full_info(object, name, value)</code>	Returns a description of the trait.
<code>get_default_value()</code>	Returns a tuple of the form: (<i>default_value_type</i> , <i>default_value</i>) which describes the default value for this trait.
<code>get_editor([trait])</code>	Returns a trait editor that allows the user to modify the <i>trait</i> trait.
<code>get_value(object, name[, trait])</code>	Returns the current value of a property-based trait.
<code>info()</code>	Must return a string describing the type of value accepted by the trait handler.
<code>init()</code>	Allows the trait to perform any additional initialization needed.
<code>inner_traits()</code>	Returns the <i>inner trait</i> (or traits) for this trait.
<code>is_valid(object, name, value)</code>	
<code>items_event()</code>	
<code>repr(value)</code>	Returns a printable representation of a value along with its type.
<code>set_value(object, name, value)</code>	Sets the cached value of a property-based trait and fires the appropriate trait change event.
<code>validate(object, name, value)</code>	

`__init__` (*trait=None*, *value=None*, *minlen=0*, *maxlen=9223372036854775807*, *items=True*, ***metadata*)

Returns a List trait.

Parameters **trait** : a trait or value that can be converted to a trait using Trait()

The type of item that the list contains. If not specified, the list can contain items of any type.

value : list

Default value for the list.

minlen : integer

The minimum length of a list that can be assigned to the trait.

maxlen : integer

The maximum length of a list that can be assigned to the trait.

The length of the list assigned to the trait must be such that:: :

$\text{minlen} \leq \text{len}(\text{list}) \leq \text{maxlen}$

as_ctrait ()

Returns a CTrait corresponding to the trait defined by this class.

clone (*default_value=<missing>*, ***metadata*)

Clones the contents of this object into a new instance of the same class, and then modifies the cloned copy using the specified *default_value* and *metadata*. Returns the cloned object as the result.

Note that subclasses can change the signature of this method if needed, but should always call the ‘super’ method if possible.

create_editor ()

Returns the default UI editor for the trait.

default_value = <undefined>

default_value_type = 5

editor = None

error (*object, name, value*)

Raises a `TraitError` exception.

Parameters **object** : object

The object whose attribute is being assigned.

name : str

The name of the attribute being assigned.

value : object

The proposed new value for the attribute.

full_info (*object, name, value*)

Returns a description of the trait.

get_default_value ()

Returns a tuple of the form: (*default_value_type, default_value*) which describes the default value for this trait. The default implementation analyzes the value of the trait's **default_value** attribute and determines an appropriate *default_value_type* for *default_value*. If you need to override this method to provide a different result tuple, the following values are valid values for *default_value_type*:

0, 1: The *default_value* item of the tuple is the default value.

2: The object containing the trait is the default value.

3: A new copy of the list specified by *default_value* is the default value.

4: A new copy of the dictionary specified by *default_value* is the default value.

5: A new instance of `TraitListObject` constructed using the *default_value* list is the default value.

6: A new instance of `TraitDictObject` constructed using the *default_value* dictionary is the default value.

7: *default_value* is a tuple of the form: (*callable, args, kw*), where *callable* is a callable, *args* is a tuple, and *kw* is either a dictionary or `None`. The default value is the result obtained by invoking `callable(*args, **kw)`.

8: *default_value* is a callable. The default value is the result obtained by invoking `default_value>(*object)`, where *object* is the object containing the trait. If the trait has a `validate()` method, the `validate()` method is also called to validate the result.

9: A new instance of `TraitSetObject` constructed using the *default_value* set is the default value.

get_editor (*trait=None*)

Returns a trait editor that allows the user to modify the *trait* trait.

Parameters **trait** : Trait

The trait to be edited.

get_value (*object, name, trait=None*)

Returns the current value of a property-based trait.

has_items = False

info ()

Must return a string describing the type of value accepted by the trait handler.

The string should be a phrase describing the type defined by the `TraitHandler` subclass, rather than a complete sentence. For example, use the phrase, “a square sprocket” instead of the sentence, “The value must be a square sprocket.” The value returned by `info()` is combined with other information whenever an error occurs and therefore makes more sense to the user if the result is a phrase. The `info()` method is similar in purpose and use to the **info** attribute of a validator function.

Note that the result can include information specific to the particular trait handler instance. For example, `TraitRange` instances return a string indicating the range of values acceptable to the handler

(e.g., “an integer in the range from 1 to 9”). If the `info()` method is not overridden, the default method returns the value of the ‘`info_text`’ attribute.

info_text = ‘a legal value’

info_trait = None

init ()

Allows the trait to perform any additional initialization needed.

inner_traits ()

Returns the *inner trait* (or traits) for this trait.

is_mapped = False

is_valid (*object*, *name*, *value*)

items_event ()

metadata = {}

repr (*value*)

Returns a printable representation of a value along with its type.

Deprecated since version 3.0.3: This functionality was only used to provide readable error messages.

This functionality has been incorporated into `TraitError` itself.

Parameters *value* : object

The value to be printed.

set_value (*object*, *name*, *value*)

Sets the cached value of a property-based trait and fires the appropriate trait change event.

validate (*object*, *name*, *value*)

NipyypeInterfaceError

class `nipyype.interfaces.base.NipyypeInterfaceError` (*value*)

Bases: `Exception`

Methods

<code>with_traceback</code>	<code>Exception.with_traceback(tb)</code>	–	<code>set</code>
	<code>self.__traceback__</code> to <code>tb</code> and return <code>self</code> .		

__init__ (*value*)

args

with_traceback ()

`Exception.with_traceback(tb)` – set `self.__traceback__` to `tb` and return `self`.

OutputMultiPath

class `nipyype.interfaces.base.OutputMultiPath` (*trait=None*, *value=None*, *minlen=0*,
maxlen=9223372036854775807,
items=True, ***metadata*)

Bases: `nipyype.interfaces.base.MultiPath`

Implements a user friendly traits that accepts one or more paths to files or directories. This is the output version which return a single string whenever possible (when it was set to a single value or a list of length 1). Default value of this trait is `_Undefined`. It does not accept empty lists.

XXX This should only be used as a final resort. We should stick to established Traits to the extent possible.

XXX This needs to be vetted by somebody who understands traits

```
>>> from nipyype.interfaces.base import OutputMultiPath
>>> class A(TraitSpec):
...     foo = OutputMultiPath(File(exists=False))
>>> a = A()
>>> a.foo
```

```
<undefined>
```

```
>>> a.foo = '/software/temp/foo.txt'
>>> a.foo
'/software/temp/foo.txt'
```

```
>>> a.foo = ['/software/temp/foo.txt']
>>> a.foo
'/software/temp/foo.txt'
```

```
>>> a.foo = ['/software/temp/foo.txt', '/software/temp/goo.txt']
>>> a.foo
['/software/temp/foo.txt', '/software/temp/goo.txt']
```

Attributes

```
editor
```

```
info_trait
```

Methods

<code>__call__(*args, **kw)</code>	Allows a derivative trait to be defined from this one.
<code>as_ctrait()</code>	Returns a CTrait corresponding to the trait defined by this class.
<code>clone([default_value])</code>	Clones the contents of this object into a new instance of the same class, and then modifies the cloned copy using the specified <i>default_value</i> and <i>metadata</i> .
<code>create_editor()</code>	Returns the default UI editor for the trait.
<code>error(object, name, value)</code>	Raises a TraitError exception.
<code>full_info(object, name, value)</code>	Returns a description of the trait.
<code>get(object, name)</code>	
<code>get_default_value()</code>	Returns a tuple of the form: (<i>default_value_type</i> , <i>default_value</i>) which describes the default value for this trait.
<code>get_editor([trait])</code>	Returns a trait editor that allows the user to modify the <i>trait</i> trait.
<code>get_value(object, name[, trait])</code>	Returns the current value of a property-based trait.
<code>info()</code>	Must return a string describing the type of value accepted by the trait handler.
<code>init()</code>	Allows the trait to perform any additional initialization needed.
<code>inner_traits()</code>	Returns the <i>inner trait</i> (or traits) for this trait.
<code>is_valid(object, name, value)</code>	
<code>items_event()</code>	
<code>repr(value)</code>	Returns a printable representation of a value along with its type.
<code>set(object, name, value)</code>	
<code>set_value(object, name, value)</code>	Sets the cached value of a property-based trait and fires the appropriate trait change event.

Continued on next page

Table 3.24 – continued from previous page

validate(object, name, value)

__init__ (*trait=None, value=None, minlen=0, maxlen=9223372036854775807, items=True, **metadata*)

Returns a List trait.

Parameters **trait** : a trait or value that can be converted to a trait using Trait()

The type of item that the list contains. If not specified, the list can contain items of any type.

value : list

Default value for the list.

minlen : integer

The minimum length of a list that can be assigned to the trait.

maxlen : integer

The maximum length of a list that can be assigned to the trait.

The length of the list assigned to the trait must be such that:: :

$\text{minlen} \leq \text{len}(\text{list}) \leq \text{maxlen}$

as_ctrait ()

Returns a CTrait corresponding to the trait defined by this class.

clone (*default_value=<missing>, **metadata*)

Clones the contents of this object into a new instance of the same class, and then modifies the cloned copy using the specified *default_value* and *metadata*. Returns the cloned object as the result.

Note that subclasses can change the signature of this method if needed, but should always call the ‘super’ method if possible.

create_editor ()

Returns the default UI editor for the trait.

default_value = <undefined>

default_value_type = 5

editor = None

error (*object, name, value*)

Raises a TraitError exception.

Parameters **object** : object

The object whose attribute is being assigned.

name : str

The name of the attribute being assigned.

value : object

The proposed new value for the attribute.

full_info (*object, name, value*)

Returns a description of the trait.

get (*object, name*)

get_default_value ()

Returns a tuple of the form: (*default_value_type, default_value*) which describes the default value for this trait. The default implementation analyzes the value of the trait’s **default_value** attribute and determines an appropriate *default_value_type* for *default_value*. If you need to override this method to provide a different result tuple, the following values are valid values for *default_value_type*:

0, 1: The *default_value* item of the tuple is the default value.

2: The object containing the trait is the default value.

3: A new copy of the list specified by *default_value* is the default value.

4: A new copy of the dictionary specified by *default_value* is the default value.

5: A new instance of `TraitListObject` constructed using the *default_value* list is the default value.

6: A new instance of `TraitDictObject` constructed using the *default_value* dictionary is the default value.

7: *default_value* is a tuple of the form: (*callable*, *args*, *kw*), where *callable* is a callable, *args* is a tuple, and *kw* is either a dictionary or `None`. The default value is the result obtained by invoking `callable(*args, **kw)`.

8: *default_value* is a callable. The default value is the result obtained by invoking *default_value**(*object*), where *object* is the object containing the trait. If the trait has a `validate()` method, the `validate()` method is also called to validate the result.

9: A new instance of `TraitSetObject` constructed using the *default_value* set is the default value.

get_editor (*trait=None*)

Returns a trait editor that allows the user to modify the *trait* trait.

Parameters *trait* : Trait

The trait to be edited.

get_value (*object*, *name*, *trait=None*)

Returns the current value of a property-based trait.

has_items = False

info ()

Must return a string describing the type of value accepted by the trait handler.

The string should be a phrase describing the type defined by the `TraitHandler` subclass, rather than a complete sentence. For example, use the phrase, “a square sprocket” instead of the sentence, “The value must be a square sprocket.” The value returned by `info()` is combined with other information whenever an error occurs and therefore makes more sense to the user if the result is a phrase. The `info()` method is similar in purpose and use to the **info** attribute of a validator function.

Note that the result can include information specific to the particular trait handler instance. For example, `TraitRange` instances return a string indicating the range of values acceptable to the handler (e.g., “an integer in the range from 1 to 9”). If the `info()` method is not overridden, the default method returns the value of the ‘`info_text`’ attribute.

info_text = ‘a legal value’

info_trait = None

init ()

Allows the trait to perform any additional initialization needed.

inner_traits ()

Returns the *inner trait* (or traits) for this trait.

is_mapped = False

is_valid (*object*, *name*, *value*)

items_event ()

metadata = {}

repr (*value*)

Returns a printable representation of a value along with its type.

Deprecated since version 3.0.3: This functionality was only used to provide readable error messages. This functionality has been incorporated into `TraitError` itself.

Parameters *value* : object

The value to be printed.

set (*object*, *name*, *value*)

set_value (*object*, *name*, *value*)

Sets the cached value of a property-based trait and fires the appropriate trait change event.

validate (*object*, *name*, *value*)

SEMLikeCommandLine

class `nipyte.interfaces.base.SEMLikeCommandLine` (*command=None, **inputs*)

Bases: `nipyte.interfaces.base.CommandLine`

In SEM derived interface all outputs have corresponding inputs. However, some SEM commands create outputs that are not defined in the XML. In those cases one has to create a subclass of the autogenerated one and overload the `_list_outputs` method. `_outputs_from_inputs` should still be used but only for the reduced (by excluding those that do not have corresponding inputs list of outputs).

Attributes

`output_spec`

Methods

<code>aggregate_outputs</code> (<i>runtime,</i> <i>needed_outputs</i>)	Collate expected outputs and check for existence
<code>help</code> (<i>[returnhelp]</i>)	
<code>input_spec</code>	alias of <code>CommandLineInputSpec</code>
<code>load_inputs_from_json</code> (<i>json_file[,</i> <i>write]</i>)	over- A convenient way to load pre-set inputs from a JSON file.
<code>raise_exception</code> (<i>runtime</i>)	
<code>run</code> (<i>**inputs</i>)	Execute this interface.
<code>save_inputs_to_json</code> (<i>json_file</i>)	A convenient way to save current inputs to a JSON file.
<code>set_default_terminal_output</code> (<i>output_type</i>)	Set the default terminal output for CommandLine Interfaces.
<code>version_from_command</code> (<i>[flag]</i>)	

`__init__` (*command=None, **inputs*)

`aggregate_outputs` (*runtime=None, needed_outputs=None*)

Collate expected outputs and check for existence

`always_run`

`can_resume`

`cmd`

sets base command, immutable

`cmdline`

command plus any arguments (*args*) validates arguments and generates command line

`help` (*returnhelp=False*)

`input_spec`

alias of `CommandLineInputSpec`

`load_inputs_from_json` (*json_file, overwrite=True*)

A convenient way to load pre-set inputs from a JSON file.

`output_spec = None`

`raise_exception` (*runtime*)

`references_` = []

`run` (***inputs*)

Execute this interface.

This interface will not raise an exception if `runtime.returncode` is non-zero.

Parameters **`inputs`** : allows the interface settings to be updated

Returns **`results`** : an `InterfaceResult` object containing a copy of the instance that was executed, provenance information and, if successful, results :

save_inputs_to_json (*json_file*)

A convenient way to save current inputs to a JSON file.

set_default_terminal_output (*output_type*)

Set the default terminal output for CommandLine Interfaces.

This method is used to set default terminal output for CommandLine Interfaces. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.terminal_output.

version

version_from_command (*flag*='-v')

StdOutCommandLine

class nipyte.interfaces.base.**StdOutCommandLine** (*command=None, **inputs*)

Bases: *nipyte.interfaces.base.CommandLine*

Attributes

output_spec

Methods

<i>aggregate_outputs</i> ([runtime, needed_outputs])	Collate expected outputs and check for existence
<i>help</i> ([returnhelp])	
<i>input_spec</i>	alias of <i>StdOutCommandLineInputSpec</i>
<i>load_inputs_from_json</i> (<i>json_file</i> [, write])	over- A convenient way to load pre-set inputs from a JSON file.
<i>raise_exception</i> (runtime)	
<i>run</i> (**inputs)	Execute this interface.
<i>save_inputs_to_json</i> (<i>json_file</i>)	A convenient way to save current inputs to a JSON file.
<i>set_default_terminal_output</i> (<i>output_type</i>)	Set the default terminal output for CommandLine Interfaces.
<i>version_from_command</i> ([flag])	

__init__ (*command=None, **inputs*)

aggregate_outputs (*runtime=None, needed_outputs=None*)

Collate expected outputs and check for existence

always_run

can_resume

cmd

sets base command, immutable

cmdline

command plus any arguments (*args*) validates arguments and generates command line

help (*returnhelp=False*)

input_spec

alias of *StdOutCommandLineInputSpec*

load_inputs_from_json (*json_file, overwrite=True*)

A convenient way to load pre-set inputs from a JSON file.

output_spec = None

raise_exception (*runtime*)

references_ = []

run (***inputs*)

Execute this interface.

This interface will not raise an exception if runtime.returncode is non-zero.

Parameters **inputs** : allows the interface settings to be updated

Returns **results** : an InterfaceResult object containing a copy of the instance

that was executed, provenance information and, if successful, results :

save_inputs_to_json (*json_file*)

A convenient way to save current inputs to a JSON file.

set_default_terminal_output (*output_type*)

Set the default terminal output for CommandLine Interfaces.

This method is used to set default terminal output for CommandLine Interfaces. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.terminal_output.

version

version_from_command (*flag='-v'*)

StdOutCommandLineInputSpec

class nipyre.interfaces.base.StdOutCommandLineInputSpec (***kwargs*)

Bases: *nipyre.interfaces.base.CommandLineInputSpec*

Methods

<i>add_class_trait</i> (name, *trait)	Adds a named trait attribute to this class.
<i>add_trait</i> (name, *trait)	Adds a trait attribute to this object.
<i>add_trait_category</i> (category)	Adds a trait category to a class.
<i>add_trait_listener</i> (object[, prefix])	
<i>all_trait_names</i> ()	Returns the list of all trait names, including implicitly defined traits.
<i>base_trait</i> (name)	Returns the base trait definition for a trait attribute.
<i>class_default_traits_view</i> ()	Returns the name of the default traits view for the class.
<i>class_editable_traits</i> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<i>class_trait_names</i> (**metadata)	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<i>class_trait_view</i> ([name, view_element])	
<i>class_trait_view_elements</i> ()	Returns the ViewElements object associated with the class.
<i>class_traits</i> (**metadata)	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
<i>class_visible_traits</i> ()	Returns an alphabetically sorted list of the names of non-event
<i>clone_traits</i> ([traits, memo, copy])	Clones a new object from this one, optionally copying only a specified set of traits.
<i>configure_traits</i> ([filename, view, kind, ...])	Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
<i>copy_traits</i> (other[, traits, memo, copy])	Copies another object's trait attributes into this one.

Continued on next page

Table 3.29 – continued from previous page

<code>copyable_trait_names(**metadata)</code>	Returns the list of trait names to copy or clone by default.
<code>default_traits_view()</code>	Returns the name of the default traits view for the object’s class.
<code>edit_traits([view, parent, kind, context, ...])</code>	Displays a user interface window for editing trait attribute values.
<code>editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.
<code>get(**kwargs)</code>	Returns traitled class as a dict
<code>get_hashval([hash_method])</code>	Return a dictionary of our items with hashes for each file.
<code>get_traitsfree(**kwargs)</code>	Returns traitled class as a dict
<code>has_metadata(name, metadata[, value, recursive])</code>	Return has_metadata for the requested trait name in this
<code>has_traits_interface(*interfaces)</code>	Returns whether the object implements a specified traits interface.
<code>items()</code>	Name, trait generator for user modifiable traits
<code>on_trait_change(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>on_trait_event(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>print_traits([show_help])</code>	Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.
<code>remove_trait(name)</code>	Removes a trait attribute from this object.
<code>remove_trait_listener(object[, prefix])</code>	
<code>reset_traits([traits])</code>	Resets some or all of an object’s trait attributes to their default values.
<code>set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>set_trait_dispatch_handler(name, klass[, ...])</code>	Sets a trait notification dispatch handler.
<code>sync_trait(trait_name, object[, alias, ...])</code>	Synchronizes the value of a trait attribute on this object with a trait attribute on another object.
<code>trait(name[, force, copy])</code>	Returns the trait definition for the <i>name</i> trait attribute.
<code>trait_context()</code>	Returns the default context to use for editing or configuring traits.
<code>trait_get(*names, **metadata)</code>	Shortcut for getting object trait attributes.
<code>trait_items_event(event_trait, name, items_event)</code>	
<code>trait_monitor(handler[, remove])</code>	Adds or removes the specified <i>handler</i> from the list of active monitors.
<code>trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>trait_property_changed(...)</code>	
<code>trait_set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>trait_setq(**traits)</code>	Shortcut for setting object trait attributes.
<code>trait_subclasses([all])</code>	Returns a list of the immediate (or all) subclasses of this class.

Continued on next page

Table 3.29 – continued from previous page

<code>trait_view([name, view_element])</code>	Gets or sets a ViewElement associated with an object’s class.
<code>trait_view_elements()</code>	Returns the ViewElements object associated with the object’s class.
<code>trait_views([klass])</code>	Returns a list of the names of all view elements associated with the current object’s class.
<code>traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of <i>metadata</i> criteria.
<code>traits_init()</code>	
<code>traits_inited([True])</code>	
<code>validate_trait(name, value)</code>	Validates whether a value is legal for a trait.
<code>visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event

__init__ (**kwargs)

Initialize handlers and inputs

add_class_trait (name, *trait)

Adds a named trait attribute to this class.

Parameters name : str

Name of the attribute to add.

*trait :

A trait or a value that can be converted to a trait using Trait() Trait definition of the attribute. It can be a single value or a list equivalent to an argument list for the Trait() function.

add_trait (name, *trait)

Adds a trait attribute to this object.

Parameters name : str

Name of the attribute to add.

*trait :

Trait or a value that can be converted to a trait by Trait(). Trait definition for name. If more than one value is specified, it is equivalent to passing the entire list of values to Trait().

add_trait_category (category)

Adds a trait category to a class.

add_trait_listener (object, prefix='')

all_trait_names ()

Returns the list of all trait names, including implicitly defined traits.

base_trait (name)

Returns the base trait definition for a trait attribute.

Parameters name : str

Name of the attribute whose trait definition is returned.

class_default_traits_view ()

Returns the name of the default traits view for the class.

class_editable_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.

class_trait_names (**metadata)

Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters **metadata :

Criteria for selecting trait attributes.

class_trait_view (name=None, view_element=None)

class_trait_view_elements()

Returns the ViewElements object associated with the class.

The returned object can be used to access all the view elements associated with the class.

class_traits(metadata)**

Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of *metadata* criteria.

Parameters ****metadata** :

Criteria for selecting trait attributes.

class_visible_traits()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class, that should be GUI visible

clone_traits(trails=None, memo=None, copy=None, **metadata)

Clones a new object from this one, optionally copying only a specified set of traits.

Parameters **trails** : list of strings

The list of names of the trait attributes to copy.

memo : dict

A dictionary of objects that have already been copied.

copy : str

The type of copy *deep* or *shallow* to perform on any trait that does not have explicit 'copy' metadata. A value of *None* means 'copy reference'.

Returns **new** :

The newly cloned object.

configure_traits(filename=None, view=None, kind=None, edit=True, context=None, handler=None, id='', scrollable=None, **args)

Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.

Parameters **filename** : str

The name (including path) of a file that contains a pickled representation of the current object. When this parameter is specified, the method reads the corresponding file (if it exists) to restore the saved values of the object's traits before displaying them. If the user confirms the dialog box (by clicking **OK**), the new values are written to the file. If this parameter is not specified, the values are loaded from the in-memory object, and are not persisted when the dialog box is closed.

view : View or str

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by *trait_view()* is used.

kind : str

The type of user interface window to create. See the **traitsui.view.kind_trait** trait for values and their meanings. If *kind* is unspecified or *None*, the **kind** attribute of the View object is used.

edit : bool

Indicates whether to display a user interface. If *filename* specifies an existing file, setting *edit* to *False* loads the saved values from that file into the object without requiring user interaction.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used

handler : Handler

A handler object used for event handling in the dialog box. If *None*, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and

position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns True on success. :

copy_traits (*other, traits=None, memo=None, copy=None, **metadata*)

Copies another object's trait attributes into this one.

Parameters other : object

The object whose trait attribute values should be copied.

traits : list of strings

A list of names of trait attributes to copy. If None or unspecified, the set of names returned by `trait_names()` is used. If 'all' or an empty list, the set of names returned by `all_trait_names()` is used.

memo : dict

A dictionary of objects that have already been copied.

copy : None | 'deep' | 'shallow'

The type of copy to perform on any trait that does not have explicit 'copy' metadata. A value of None means 'copy reference'.

Returns unassignable : list of strings

A list of attributes that the method was unable to copy, which is empty if all the attributes were successfully copied.

copyable_trait_names (***metadata*)

Returns the list of trait names to copy or clone by default.

default_traits_view ()

Returns the name of the default traits view for the object's class.

edit_traits (*view=None, parent=None, kind=None, context=None, handler=None, id='', scrollable=None, **args*)

Displays a user interface window for editing trait attribute values.

Parameters view : View or string

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

parent : toolkit control

The reference to a user interface component to use as the parent window for the object's UI window.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or None, the **kind** attribute of the View object is used.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used.

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns A UI object. :

editable_traits()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

get (***kwargs*)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without notification handles

get_hashval (*hash_method=None*)

Return a dictionary of our items with hashes for each file.

Searches through dictionary items and if an item is a file, it calculates the md5 hash of the file contents and stores the file name and hash value as the new key value.

However, the overall bunch hash is calculated only on the hash value of a file. The path and name of the file are not used in the overall hash calculation.

Returns dict_withhash : dict

Copy of our dictionary with the new file hashes included with each file.

hashvalue : str

The md5 hash value of the traitled spec

get_traitsfree (***kwargs*)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without any traits. The dictionary does not contain any attributes that were Undefined

has_metadata (*name, metadata, value=None, recursive=True*)

Return has_metadata for the requested trait name in this interface

has_traits_interface (**interfaces*)

Returns whether the object implements a specified traits interface.

Parameters **interfaces* : :

One or more traits Interface (sub)classes.

items()

Name, trait generator for user modifiable traits

on_trait_change (*handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters *handler* : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

on_trait_event (*handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters **handler** : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

print_traits (*show_help=False, **metadata*)

Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.

Parameters **show_help** : bool

Indicates whether to display additional descriptive information.

remove_trait (*name*)

Removes a trait attribute from this object.

Parameters **name** : str

Name of the attribute to remove.

Returns **result** : bool

True if the trait was successfully removed.

remove_trait_listener (*object, prefix=''*)

reset_traits (*traits=None, **metadata*)

Resets some or all of an object's trait attributes to their default values.

Parameters **traits** : list of strings

Names of trait attributes to reset.

Returns **unresetable** : list of strings

A list of attributes that the method was unable to reset, which is empty if all the attributes were successfully reset.

set (*trait_change_notify=True, **traits*)

Shortcut for setting object trait attributes.

Parameters **trait_change_notify** : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: *trait_setq*)

****traits** :

Key/value pairs, the trait attributes and their values to be set

Returns **self** :

The method returns this object, after setting attributes.

set_trait_dispatch_handler (*name, klass, override=False*)

Sets a trait notification dispatch handler.

sync_trait (*trait_name, object, alias=None, mutual=True, remove=False*)

Synchronizes the value of a trait attribute on this object with a trait attribute on another object.

Parameters **name** : str

Name of the trait attribute on this object.

object : object
The object with which to synchronize.

alias : str
Name of the trait attribute on *other*; if None or omitted, same as *name*.

mutual : bool or int
Indicates whether synchronization is mutual (True or non-zero) or one-way (False or zero)

remove : bool or int
Indicates whether synchronization is being added (False or zero) or removed (True or non-zero)

trait (*name*, *force=False*, *copy=False*)
Returns the trait definition for the *name* trait attribute.

Parameters **name** : str
Name of the attribute whose trait definition is to be returned.

force : bool
Indicates whether to return a trait definition if *name* is not explicitly defined.

copy : bool
Indicates whether to return the original trait definition or a copy.

trait_context ()
Returns the default context to use for editing or configuring traits.

trait_get (**names*, ***metadata*)
Shortcut for getting object trait attributes.

Parameters **names** : list of strings
A list of trait attribute names whose values are requested.

Returns **result** : dict
A dictionary whose keys are the names passed as arguments and whose values are the corresponding trait values.

trait_items_event (*event_trait*, *name*, *items_event*)

trait_monitor (*handler*, *remove=False*)
Adds or removes the specified *handler* from the list of active monitors.

Parameters **handler** : function
The function to add or remove as a monitor.

remove : bool
Flag indicating whether to remove (True) or add the specified handler as a monitor for this class.

trait_names (***metadata*)
Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters ****metadata** : :
Criteria for selecting trait attributes.

trait_property_changed (*name*, *old_value* [, *new_value*])

trait_set (*trait_change_notify=True*, ***traits*)
Shortcut for setting object trait attributes.

Parameters **trait_change_notify** : bool
If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: *trait_setq*)

****traits** : :
Key/value pairs, the trait attributes and their values to be set

Returns **self** : :
The method returns this object, after setting attributes.

trait_setq (***traits*)
Shortcut for setting object trait attributes.

Parameters ****traits** : :

Key/value pairs, the trait attributes and their values to be set. No trait change notifications will be generated for any values assigned (see also: `trait_set`).

Returns self :

The method returns this object, after setting attributes.

trait_subclasses (*all=False*)

Returns a list of the immediate (or all) subclasses of this class.

Parameters all : bool

Indicates whether to return all subclasses of this class. If False, only immediate subclasses are returned.

trait_view (*name=None, view_element=None*)

Gets or sets a ViewElement associated with an object's class.

Parameters name : str

Name of a view element

view_element : ViewElement

View element to associate

Returns A view element. :

trait_view_elements ()

Returns the ViewElements object associated with the object's class.

The returned object can be used to access all the view elements associated with the class.

trait_views (*klass=None*)

Returns a list of the names of all view elements associated with the current object's class.

Parameters klass : class

A class, such that all returned names must correspond to instances of this class. Possible values include:

Group

Item

View

ViewElement

ViewSubElement

traits (***metadata*)

Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of *metadata* criteria.

Parameters **metadata :

Criteria for selecting trait attributes.

traits_init ()

traits_initiated ([True])

validate_trait (*name, value*)

Validates whether a value is legal for a trait.

Returns the validated value if it is valid.

visible_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object, that should be GUI visible

wrappers = {'ui': <class 'traits.trait_notifiers.FastUITraitChangeNotifyWrapper'>, 'new': <class 'traits.trait_notifiers.NewWrapper'>}

Str

class nipytype.interfaces.base.Str (*default_value=<traits.trait_handlers.NoDefaultSpecified object>, **metadata*)

Bases: traits.trait_types.Unicode

Attributes

Continued on next page

Table 3.30 – continued from previous page

<i>editor</i>	
Methods	
<code>__call__(*args, **kw)</code>	Allows a derivative trait to be defined from this one.
<code>as_ctrait()</code>	Returns a CTrait corresponding to the trait defined by this class.
<code>clone([default_value])</code>	Clones the contents of this object into a new instance of the same class, and then modifies the cloned copy using the specified <i>default_value</i> and <i>metadata</i> .
<code>create_editor()</code>	Returns the default traits UI editor for this type of trait.
<code>error(object, name, value)</code>	Raises a TraitError exception.
<code>full_info(object, name, value)</code>	Returns a string describing the type of value accepted by the trait handler.
<code>get_default_value()</code>	Returns a tuple of the form: (<i>default_value_type</i> , <i>default_value</i>) which describes the default value for this trait.
<code>get_editor([trait])</code>	Returns a trait editor that allows the user to modify the <i>trait</i> trait.
<code>get_value(object, name[, trait])</code>	Returns the current value of a property-based trait.
<code>info()</code>	Must return a string describing the type of value accepted by the trait handler.
<code>init()</code>	Allows the trait to perform any additional initialization needed.
<code>inner_traits()</code>	Returns a tuple containing the <i>inner traits</i> for this trait.
<code>is_valid(object, name, value)</code>	
<code>repr(value)</code>	Returns a printable representation of a value along with its type.
<code>set_value(object, name, value)</code>	Sets the cached value of a property-based trait and fires the appropriate trait change event.
<code>validate(object, name, value)</code>	Validates that a specified value is valid for this trait.

__init__ (*default_value*=<traits.trait_handlers.NoDefaultSpecified object>, ***metadata*)

This constructor method is the only method normally called directly by client code. It defines the trait. The default implementation accepts an optional, untype-checked default value, and caller-supplied trait metadata. Override this method whenever a different method signature or a type-checked default value is needed.

as_ctrait ()

Returns a CTrait corresponding to the trait defined by this class.

clone (*default_value*=<missing>, ***metadata*)

Clones the contents of this object into a new instance of the same class, and then modifies the cloned copy using the specified *default_value* and *metadata*. Returns the cloned object as the result.

Note that subclasses can change the signature of this method if needed, but should always call the ‘super’ method if possible.

create_editor ()

Returns the default traits UI editor for this type of trait.

default_value = ‘

default_value_type = -1

editor = None

error (*object*, *name*, *value*)

Raises a TraitError exception.

Parameters *object* : object

The object whose attribute is being assigned.

name : str

The name of the attribute being assigned.

value : object

The proposed new value for the attribute.

fast_validate = (11, <class 'str'>, None, <class 'str'>)

full_info (*object*, *name*, *value*)

Returns a string describing the type of value accepted by the trait handler.

Parameters *object* : object

The object whose attribute is being assigned.

name : str

The name of the attribute being assigned.

value :

The proposed new value for the attribute.

get_default_value ()

Returns a tuple of the form: (*default_value_type*, *default_value*) which describes the default value for this trait. The default implementation analyzes the value of the trait's **default_value** attribute and determines an appropriate *default_value_type* for *default_value*. If you need to override this method to provide a different result tuple, the following values are valid values for *default_value_type*:

0, 1: The *default_value* item of the tuple is the default value.

2: The object containing the trait is the default value.

3: A new copy of the list specified by *default_value* is the default value.

4: A new copy of the dictionary specified by *default_value* is the default value.

5: A new instance of TraitListObject constructed using the *default_value* list is the default value.

6: A new instance of TraitDictObject constructed using the *default_value* dictionary is the default value.

7: *default_value* is a tuple of the form: (*callable*, *args*, *kw*), where *callable* is a callable, *args* is a tuple, and *kw* is either a dictionary or None. The default value is the result obtained by invoking *callable*(**args*, ***kw*).

8: *default_value* is a callable. The default value is the result obtained by invoking *default_value*(**object*), where *object* is the object containing the trait. If the trait has a *validate*() method, the *validate*() method is also called to validate the result.

9: A new instance of TraitSetObject constructed using the *default_value* set is the default value.

get_editor (*trait*=None)

Returns a trait editor that allows the user to modify the *trait* trait.

Parameters *trait* : Trait

The trait to be edited.

get_value (*object*, *name*, *trait*=None)

Returns the current value of a property-based trait.

has_items = False

info ()

Must return a string describing the type of value accepted by the trait handler.

The string should be a phrase describing the type defined by the `TraitHandler` subclass, rather than a complete sentence. For example, use the phrase, “a square sprocket” instead of the sentence, “The value must be a square sprocket.” The value returned by `info()` is combined with other information whenever an error occurs and therefore makes more sense to the user if the result is a phrase. The `info()` method is similar in purpose and use to the **info** attribute of a validator function.

Note that the result can include information specific to the particular trait handler instance. For example, `TraitRange` instances return a string indicating the range of values acceptable to the handler (e.g., “an integer in the range from 1 to 9”). If the `info()` method is not overridden, the default method returns the value of the ‘info_text’ attribute.

info_text = ‘a unicode string’

init ()

Allows the trait to perform any additional initialization needed.

inner_traits ()

Returns a tuple containing the *inner traits* for this trait. Most trait handlers do not have any inner traits, and so will return an empty tuple. The exceptions are **List** and **Dict** trait types, which have inner traits used to validate the values assigned to the trait. For example, in `List(Int)`, the *inner traits* for **List** are (**Int**,).

is_mapped = False

is_valid (object, name, value)

metadata = {}

repr (value)

Returns a printable representation of a value along with its type.

Deprecated since version 3.0.3: This functionality was only used to provide readable error messages.

This functionality has been incorporated into `TraitError` itself.

Parameters value : object

The value to be printed.

set_value (object, name, value)

Sets the cached value of a property-based trait and fires the appropriate trait change event.

validate (object, name, value)

Validates that a specified value is valid for this trait.

Note: The ‘fast validator’ version performs this check in C.

Stream

class `nipyte.interfaces.base.Stream` (name, impl)

Bases: object

Function to capture stdout and stderr streams with timestamps

stackoverflow.com/questions/4984549/merge-and-sync-stdout-and-stderr/5188359

Methods

<code>fileno()</code>	Pass-through for file descriptor.
<code>read([drain])</code>	Read from the file descriptor.

__init__ (name, impl)

fileno ()

Pass-through for file descriptor.

read (drain=0)

Read from the file descriptor. If ‘drain’ set, read until EOF.

TraitedSpec

class `nipy.interfaces.base.TraitedSpec` (***kwargs*)
 Bases: `nipy.interfaces.base.BaseTraitedSpec`
 Create a subclass with strict traits.
 This is used in 90% of the cases.

Methods

<code>add_class_trait(name, *trait)</code>	Adds a named trait attribute to this class.
<code>add_trait(name, *trait)</code>	Adds a trait attribute to this object.
<code>add_trait_category(category)</code>	Adds a trait category to a class.
<code>add_trait_listener(object[, prefix])</code>	
<code>all_trait_names()</code>	Returns the list of all trait names, including implicitly defined traits.
<code>base_trait(name)</code>	Returns the base trait definition for a trait attribute.
<code>class_default_traits_view()</code>	Returns the name of the default traits view for the class.
<code>class_editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<code>class_trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>class_trait_view([name, view_element])</code>	
<code>class_trait_view_elements()</code>	Returns the ViewElements object associated with the class.
<code>class_traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
<code>class_visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event
<code>clone_traits([traits, memo, copy])</code>	Clones a new object from this one, optionally copying only a specified set of traits.
<code>configure_traits([filename, view, kind, ...])</code>	Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
<code>copy_traits(other[, traits, memo, copy])</code>	Copies another object's trait attributes into this one.
<code>copyable_trait_names(**metadata)</code>	Returns the list of trait names to copy or clone by default.
<code>default_traits_view()</code>	Returns the name of the default traits view for the object's class.
<code>edit_traits([view, parent, kind, context, ...])</code>	Displays a user interface window for editing trait attribute values.
<code>editable_traits()</code>	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.
<code>get(**kwargs)</code>	Returns traied class as a dict
<code>get_hashval([hash_method])</code>	Return a dictionary of our items with hashes for each file.
<code>get_traitsfree(**kwargs)</code>	Returns traied class as a dict

Continued on next page

Table 3.33 – continued from previous page

<code>has_metadata(name, metadata[, value, recursive])</code>	Return <code>has_metadata</code> for the requested trait name in this
<code>has_traits_interface(*interfaces)</code>	Returns whether the object implements a specified traits interface.
<code>items()</code>	Name, trait generator for user modifiable traits
<code>on_trait_change(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>on_trait_event(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>print_traits([show_help])</code>	Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.
<code>remove_trait(name)</code>	Removes a trait attribute from this object.
<code>remove_trait_listener(object[, prefix])</code>	
<code>reset_traits([traits])</code>	Resets some or all of an object's trait attributes to their default values.
<code>set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>set_trait_dispatch_handler(name, klass[, ...])</code>	Sets a trait notification dispatch handler.
<code>sync_trait(trait_name, object[, alias, ...])</code>	Synchronizes the value of a trait attribute on this object with a trait attribute on another object.
<code>trait(name[, force, copy])</code>	Returns the trait definition for the <i>name</i> trait attribute.
<code>trait_context()</code>	Returns the default context to use for editing or configuring traits.
<code>trait_get(*names, **metadata)</code>	Shortcut for getting object trait attributes.
<code>trait_items_event(event_trait, name, items_event)</code>	
<code>trait_monitor(handler[, remove])</code>	Adds or removes the specified <i>handler</i> from the list of active monitors.
<code>trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>trait_property_changed(...)</code>	
<code>trait_set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>trait_setq(**traits)</code>	Shortcut for setting object trait attributes.
<code>trait_subclasses([all])</code>	Returns a list of the immediate (or all) subclasses of this class.
<code>trait_view([name, view_element])</code>	Gets or sets a <code>ViewElement</code> associated with an object's class.
<code>trait_view_elements()</code>	Returns the <code>ViewElements</code> object associated with the object's class.
<code>trait_views([klass])</code>	Returns a list of the names of all view elements associated with the current object's class.
<code>traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of <i>metadata</i> criteria.
<code>traits_init()</code>	
<code>traits_inited([True])</code>	
<code>validate_trait(name, value)</code>	Validates whether a value is legal for a trait.

Continued on next page

Table 3.33 – continued from previous page

<code>visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event
<hr/>	
<code>__init__</code> (<i>**kwargs</i>)	Initialize handlers and inputs
<code>add_class_trait</code> (<i>name</i> , <i>*trait</i>)	Adds a named trait attribute to this class.
Parameters <i>name</i> : str	Name of the attribute to add.
<i>*trait</i> :	A trait or a value that can be converted to a trait using <code>Trait()</code> . Trait definition of the attribute. It can be a single value or a list equivalent to an argument list for the <code>Trait()</code> function.
<code>add_trait</code> (<i>name</i> , <i>*trait</i>)	Adds a trait attribute to this object.
Parameters <i>name</i> : str	Name of the attribute to add.
<i>*trait</i> :	Trait or a value that can be converted to a trait by <code>Trait()</code> . Trait definition for <i>name</i> . If more than one value is specified, it is equivalent to passing the entire list of values to <code>Trait()</code> .
<code>add_trait_category</code> (<i>category</i>)	Adds a trait category to a class.
<code>add_trait_listener</code> (<i>object</i> , <i>prefix</i> = <i>''</i>)	
<code>all_trait_names</code> ()	Returns the list of all trait names, including implicitly defined traits.
<code>base_trait</code> (<i>name</i>)	Returns the base trait definition for a trait attribute.
Parameters <i>name</i> : str	Name of the attribute whose trait definition is returned.
<code>class_default_traits_view</code> ()	Returns the name of the default traits view for the class.
<code>class_editable_traits</code> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<code>class_trait_names</code> (<i>**metadata</i>)	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
Parameters <i>**metadata</i> :	Criteria for selecting trait attributes.
<code>class_trait_view</code> (<i>name</i> = <i>None</i> , <i>view_element</i> = <i>None</i>)	
<code>class_trait_view_elements</code> ()	Returns the <code>ViewElements</code> object associated with the class.
	The returned object can be used to access all the view elements associated with the class.
<code>class_traits</code> (<i>**metadata</i>)	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
Parameters <i>**metadata</i> :	Criteria for selecting trait attributes.
<code>class_visible_traits</code> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class, that should be GUI visible
<code>clone_traits</code> (<i>traits</i> = <i>None</i> , <i>memo</i> = <i>None</i> , <i>copy</i> = <i>None</i> , <i>**metadata</i>)	

Clones a new object from this one, optionally copying only a specified set of traits.

Parameters **traits** : list of strings

The list of names of the trait attributes to copy.

memo : dict

A dictionary of objects that have already been copied.

copy : str

The type of copy `deep` or `shallow` to perform on any trait that does not have explicit 'copy' metadata. A value of `None` means 'copy reference'.

Returns **new** :

The newly cloned object.

configure_traits (*filename=None, view=None, kind=None, edit=True, context=None, handler=None, id='', scrollable=None, **args*)

Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.

Parameters **filename** : str

The name (including path) of a file that contains a pickled representation of the current object. When this parameter is specified, the method reads the corresponding file (if it exists) to restore the saved values of the object's traits before displaying them. If the user confirms the dialog box (by clicking **OK**), the new values are written to the file. If this parameter is not specified, the values are loaded from the in-memory object, and are not persisted when the dialog box is closed.

view : View or str

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

kind : str

The type of user interface window to create. See the **traitsui.view.kind_trait** trait for values and their meanings. If *kind* is unspecified or `None`, the **kind** attribute of the View object is used.

edit : bool

Indicates whether to display a user interface. If *filename* specifies an existing file, setting *edit* to `False` loads the saved values from that file into the object without requiring user interaction.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used

handler : Handler

A handler object used for event handling in the dialog box. If `None`, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to `True`, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns **True on success.** :

copy_traits (*other, traits=None, memo=None, copy=None, **metadata*)

Copies another object's trait attributes into this one.

Parameters **other** : object

The object whose trait attribute values should be copied.

traits : list of strings

A list of names of trait attributes to copy. If `None` or unspecified, the set of names

returned by `trait_names()` is used. If 'all' or an empty list, the set of names returned by `all_trait_names()` is used.

memo : dict

A dictionary of objects that have already been copied.

copy : None | 'deep' | 'shallow'

The type of copy to perform on any trait that does not have explicit 'copy' metadata.

A value of None means 'copy reference'.

Returns unassignable : list of strings

A list of attributes that the method was unable to copy, which is empty if all the attributes were successfully copied.

copyable_trait_names (***metadata*)

Returns the list of trait names to copy or clone by default.

default_traits_view ()

Returns the name of the default traits view for the object's class.

edit_traits (*view=None, parent=None, kind=None, context=None, handler=None, id='', scrollable=None, **args*)

Displays a user interface window for editing trait attribute values.

Parameters view : View or string

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

parent : toolkit control

The reference to a user interface component to use as the parent window for the object's UI window.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or None, the **kind** attribute of the View object is used.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used.

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns A UI object. :

editable_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

get (***kwargs*)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without notification handles

get_hashval (*hash_method=None*)

Return a dictionary of our items with hashes for each file.

Searches through dictionary items and if an item is a file, it calculates the md5 hash of the file contents and stores the file name and hash value as the new key value.

However, the overall bunch hash is calculated only on the hash value of a file. The path and name of

the file are not used in the overall hash calculation.

Returns `dict_withhash` : dict

Copy of our dictionary with the new file hashes included with each file.

hashvalue : str

The md5 hash value of the traitled spec

get_traitsfree (***kwargs*)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without any traits. The dictionary does not contain any attributes that were Undefined

has_metadata (*name, metadata, value=None, recursive=True*)

Return has_metadata for the requested trait name in this interface

has_traits_interface (**interfaces*)

Returns whether the object implements a specified traits interface.

Parameters **interfaces* :

One or more traits Interface (sub)classes.

items ()

Name, trait generator for user modifiable traits

on_trait_change (*handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters *handler* : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

on_trait_event (*handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters *handler* : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

print_traits (*show_help=False, **metadata*)

Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.

Parameters **show_help** : bool

Indicates whether to display additional descriptive information.

remove_trait (*name*)

Removes a trait attribute from this object.

Parameters **name** : str

Name of the attribute to remove.

Returns **result** : bool

True if the trait was successfully removed.

remove_trait_listener (*object, prefix=''*)

reset_traits (*traits=None, **metadata*)

Resets some or all of an object's trait attributes to their default values.

Parameters **traits** : list of strings

Names of trait attributes to reset.

Returns **unresetable** : list of strings

A list of attributes that the method was unable to reset, which is empty if all the attributes were successfully reset.

set (*trait_change_notify=True, **traits*)

Shortcut for setting object trait attributes.

Parameters **trait_change_notify** : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: `trait_setq`)

****traits** :

Key/value pairs, the trait attributes and their values to be set

Returns **self** :

The method returns this object, after setting attributes.

set_trait_dispatch_handler (*name, klass, override=False*)

Sets a trait notification dispatch handler.

sync_trait (*trait_name, object, alias=None, mutual=True, remove=False*)

Synchronizes the value of a trait attribute on this object with a trait attribute on another object.

Parameters **name** : str

Name of the trait attribute on this object.

object : object

The object with which to synchronize.

alias : str

Name of the trait attribute on *other*; if None or omitted, same as *name*.

mutual : bool or int

Indicates whether synchronization is mutual (True or non-zero) or one-way (False or zero)

remove : bool or int

Indicates whether synchronization is being added (False or zero) or removed (True or non-zero)

trait (*name, force=False, copy=False*)

Returns the trait definition for the *name* trait attribute.

Parameters *name* : str

Name of the attribute whose trait definition is to be returned.

force : bool

Indicates whether to return a trait definition if *name* is not explicitly defined.

copy : bool

Indicates whether to return the original trait definition or a copy.

trait_context ()

Returns the default context to use for editing or configuring traits.

trait_get (**names*, ***metadata*)

Shortcut for getting object trait attributes.

Parameters *names* : list of strings

A list of trait attribute names whose values are requested.

Returns *result* : dict

A dictionary whose keys are the names passed as arguments and whose values are the corresponding trait values.

trait_items_event (*event_trait*, *name*, *items_event*)

trait_monitor (*handler*, *remove=False*)

Adds or removes the specified *handler* from the list of active monitors.

Parameters *handler* : function

The function to add or remove as a monitor.

remove : bool

Flag indicating whether to remove (True) or add the specified handler as a monitor for this class.

trait_names (***metadata*)

Returns a list of the names of all trait attributes whose definitions match the set of *metadata* criteria specified.

Parameters ***metadata* : :

Criteria for selecting trait attributes.

trait_property_changed (*name*, *old_value* [, *new_value*])

trait_set (*trait_change_notify=True*, ***traits*)

Shortcut for setting object trait attributes.

Parameters *trait_change_notify* : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: *trait_setq*)

****traits** : :

Key/value pairs, the trait attributes and their values to be set

Returns *self* : :

The method returns this object, after setting attributes.

trait_setq (***traits*)

Shortcut for setting object trait attributes.

Parameters ***traits* : :

Key/value pairs, the trait attributes and their values to be set. No trait change notifications will be generated for any values assigned (see also: *trait_set*).

Returns *self* : :

The method returns this object, after setting attributes.

trait_subclasses (*all=False*)

Returns a list of the immediate (or all) subclasses of this class.

Parameters *all* : bool

Indicates whether to return all subclasses of this class. If **False**, only immediate subclasses are returned.

trait_view (*name=None*, *view_element=None*)

Gets or sets a *ViewElement* associated with an object's class.

Parameters *name* : str

Name of a view element
view_element : ViewElement
 View element to associate
Returns A view element. :

trait_view_elements ()
 Returns the ViewElements object associated with the object's class.
 The returned object can be used to access all the view elements associated with the class.

trait_views (klass=None)
 Returns a list of the names of all view elements associated with the current object's class.
Parameters klass : class
 A class, such that all returned names must correspond to instances of this class. Possible values include:
 Group
 Item
 View
 ViewElement
 ViewSubElement

traits (**metadata)
 Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of *metadata* criteria.
Parameters **metadata : :
 Criteria for selecting trait attributes.

traits_init ()
traits_initiated ([True])
validate_trait (name, value)
 Validates whether a value is legal for a trait.
 Returns the validated value if it is valid.

visible_traits ()
 Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object, that should be GUI visible

wrappers = {'ui': <class 'traits.trait_notifiers.FastUITraitChangeNotifyWrapper'>, 'new': <class 'traits.trait_notifiers.NewUITraitChangeNotifyWrapper'>}

Functions

nipyype.interfaces.base.get_dependencies (name, environ)
 Return library dependencies of a dynamically linked executable
 Uses otool on darwin, ldd on linux. Currently doesn't support windows.

nipyype.interfaces.base.get_max_resources_used (pid, mem_mb, num_threads, pyfunc=False)
 Function to get the RAM and threads usage of a process
Parameters pid : integer
 the process ID of process to profile
 mem_mb : float
 the high memory watermark so far during process execution (in MB)
 num_threads: int :
 the high thread watermark so far during process execution
Returns mem_mb : float
 the new high memory watermark of process (MB)
 num_threads : float
 the new high thread watermark of process

nipyype.interfaces.base.load_template (name)
 Load a template from the script_templates directory
Parameters name : str
 The name of the file to load

Returns `template` : `string.Template`

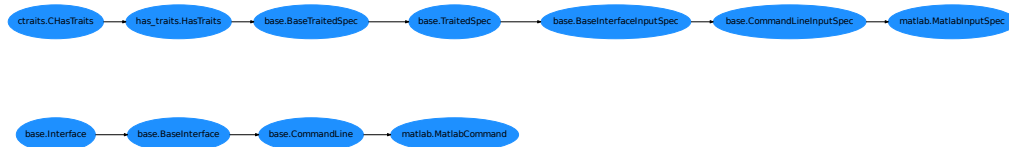
`nipyne.interfaces.base.run_command(runtime, output=None, timeout=0.01, redirect_x=False)`

Run a command, read stdout and stderr, prefix with timestamp.
The returned runtime contains a merged stdout+stderr log with timestamps

interfaces.matlab

Module: `interfaces.matlab`

Inheritance diagram for `nipyne.interfaces.matlab`:



General matlab interface code

Classes

`MatlabCommand`

class `nipyne.interfaces.matlab.MatlabCommand(matlab_cmd=None, **inputs)`
Bases: `nipyne.interfaces.base.CommandLine`
Interface that runs matlab code

```

>>> import nipyne.interfaces.matlab as matlab
>>> mlab = matlab.MatlabCommand(mfile=False) # don't write script file
>>> mlab.inputs.script = "which('who')"
>>> out = mlab.run()
  
```

Attributes

`output_spec`

Methods

<code>aggregate_outputs([runtime, needed_outputs])</code>	Collate expected outputs and check for existence
<code>help([returnhelp])</code>	
<code>input_spec</code>	alias of <code>MatlabInputSpec</code>
<code>load_inputs_from_json(json_file[, overwrite])</code>	A convenient way to load pre-set inputs from a JSON file.
<code>raise_exception(runtime)</code>	
<code>run(**inputs)</code>	Execute this interface.

Continued on next page

Table 3.35 – continued from previous page

<code>save_inputs_to_json(json_file)</code>	A convenient way to save current inputs to a JSON file.
<code>set_default_matlab_cmd(matlab_cmd)</code>	Set the default MATLAB command line for MATLAB classes.
<code>set_default_mfile(mfile)</code>	Set the default MATLAB script file format for MATLAB classes.
<code>set_default_paths(paths)</code>	Set the default MATLAB paths for MATLAB classes.
<code>set_default_terminal_output(output_type)</code>	Set the default terminal output for CommandLine Interfaces.
<code>version_from_command([flag])</code>	

__init__ (*matlab_cmd=None, **inputs*)
 initializes interface to matlab (default ‘matlab -nodesktop -nosplash’)

aggregate_outputs (*runtime=None, needed_outputs=None*)
 Collate expected outputs and check for existence

always_run

can_resume

cmd
 sets base command, immutable

cmdline
command plus any arguments (*args*) validates arguments and generates command line

help (*returnhelp=False*)

input_spec
 alias of *MatlabInputSpec*

load_inputs_from_json (*json_file, overwrite=True*)
 A convenient way to load pre-set inputs from a JSON file.

output_spec = None

raise_exception (*runtime*)

references_ = []

run (***inputs*)
 Execute this interface.
 This interface will not raise an exception if runtime.returncode is non-zero.

Parameters **inputs** : allows the interface settings to be updated

Returns **results** : an InterfaceResult object containing a copy of the instance
that was executed, provenance information and, if successful, results :

save_inputs_to_json (*json_file*)
 A convenient way to save current inputs to a JSON file.

classmethod set_default_matlab_cmd (*matlab_cmd*)
 Set the default MATLAB command line for MATLAB classes.
 This method is used to set values for all MATLAB subclasses. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.matlab_cmd.

classmethod set_default_mfile (*mfile*)
 Set the default MATLAB script file format for MATLAB classes.
 This method is used to set values for all MATLAB subclasses. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.mfile.

classmethod set_default_paths (*paths*)
 Set the default MATLAB paths for MATLAB classes.
 This method is used to set values for all MATLAB subclasses. However, setting this will not update the output type for any existing instances. For these, assign the <instance>.inputs.paths.

set_default_terminal_output (*output_type*)
 Set the default terminal output for CommandLine Interfaces.
 This method is used to set default terminal output for CommandLine Interfaces. However, setting this will not update the output type for any existing instances. For these, assign the <in-

```

    stance>.inputs.terminal_output.
version
version_from_command (flag='-v')

```

MatlabInputSpec

class `nipyne.interfaces.matlab.MatlabInputSpec` (**kwargs)
 Bases: `nipyne.interfaces.base.CommandLineInputSpec`
 Basic expected inputs to Matlab interface

Methods

<code>add_class_trait</code> (name, *trait)	Adds a named trait attribute to this class.
<code>add_trait</code> (name, *trait)	Adds a trait attribute to this object.
<code>add_trait_category</code> (category)	Adds a trait category to a class.
<code>add_trait_listener</code> (object[, prefix])	
<code>all_trait_names</code> ()	Returns the list of all trait names, including implicitly defined traits.
<code>base_trait</code> (name)	Returns the base trait definition for a trait attribute.
<code>class_default_traits_view</code> ()	Returns the name of the default traits view for the class.
<code>class_editable_traits</code> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.
<code>class_trait_names</code> (**metadata)	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>class_trait_view</code> ([name, view_element])	
<code>class_trait_view_elements</code> ()	Returns the ViewElements object associated with the class.
<code>class_traits</code> (**metadata)	Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of <i>metadata</i> criteria.
<code>class_visible_traits</code> ()	Returns an alphabetically sorted list of the names of non-event
<code>clone_traits</code> ([traits, memo, copy])	Clones a new object from this one, optionally copying only a specified set of traits.
<code>configure_traits</code> ([filename, view, kind, ...])	Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.
<code>copy_traits</code> (other[, traits, memo, copy])	Copies another object's trait attributes into this one.
<code>copyable_trait_names</code> (**metadata)	Returns the list of trait names to copy or clone by default.
<code>default_traits_view</code> ()	Returns the name of the default traits view for the object's class.
<code>edit_traits</code> ([view, parent, kind, context, ...])	Displays a user interface window for editing trait attribute values.
<code>editable_traits</code> ()	Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.
<code>get</code> (**kwargs)	Returns traitled class as a dict

Continued on next page

Table 3.36 – continued from previous page

<code>get_hashval([hash_method])</code>	Return a dictionary of our items with hashes for each file.
<code>get_traitsfree(**kwargs)</code>	Returns traited class as a dict
<code>has_metadata(name, metadata[, value, recursive])</code>	Return <code>has_metadata</code> for the requested trait name in this
<code>has_traits_interface(*interfaces)</code>	Returns whether the object implements a specified traits interface.
<code>items()</code>	Name, trait generator for user modifiable traits
<code>on_trait_change(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>on_trait_event(handler[, name, remove, ...])</code>	Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.
<code>print_traits([show_help])</code>	Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.
<code>remove_trait(name)</code>	Removes a trait attribute from this object.
<code>remove_trait_listener(object[, prefix])</code>	
<code>reset_traits([traits])</code>	Resets some or all of an object's trait attributes to their default values.
<code>set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>set_trait_dispatch_handler(name, klass[, ...])</code>	Sets a trait notification dispatch handler.
<code>sync_trait(trait_name, object[, alias, ...])</code>	Synchronizes the value of a trait attribute on this object with a trait attribute on another object.
<code>trait(name[, force, copy])</code>	Returns the trait definition for the <i>name</i> trait attribute.
<code>trait_context()</code>	Returns the default context to use for editing or configuring traits.
<code>trait_get(*names, **metadata)</code>	Shortcut for getting object trait attributes.
<code>trait_items_event(event_trait, name, items_event)</code>	
<code>trait_monitor(handler[, remove])</code>	Adds or removes the specified <i>handler</i> from the list of active monitors.
<code>trait_names(**metadata)</code>	Returns a list of the names of all trait attributes whose definitions match the set of <i>metadata</i> criteria specified.
<code>trait_property_changed(...)</code>	
<code>trait_set([trait_change_notify])</code>	Shortcut for setting object trait attributes.
<code>trait_setq(**traits)</code>	Shortcut for setting object trait attributes.
<code>trait_subclasses([all])</code>	Returns a list of the immediate (or all) subclasses of this class.
<code>trait_view([name, view_element])</code>	Gets or sets a <code>ViewElement</code> associated with an object's class.
<code>trait_view_elements()</code>	Returns the <code>ViewElements</code> object associated with the object's class.
<code>trait_views([klass])</code>	Returns a list of the names of all view elements associated with the current object's class.
<code>traits(**metadata)</code>	Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of <i>metadata</i> criteria.
<code>traits_init()</code>	

Continued on next page

Table 3.36 – continued from previous page

<code>traits_inited([True])</code>	
<code>validate_trait(name, value)</code>	Validates whether a value is legal for a trait.
<code>visible_traits()</code>	Returns an alphabetically sorted list of the names of non-event

`__init__` (**kwargs)

Initialize handlers and inputs

`add_class_trait` (name, *trait)

Adds a named trait attribute to this class.

Parameters name : str

Name of the attribute to add.

*trait : :

A trait or a value that can be converted to a trait using Trait() Trait definition of the attribute. It can be a single value or a list equivalent to an argument list for the Trait() function.

`add_trait` (name, *trait)

Adds a trait attribute to this object.

Parameters name : str

Name of the attribute to add.

*trait : :

Trait or a value that can be converted to a trait by Trait(). Trait definition for name. If more than one value is specified, it is equivalent to passing the entire list of values to Trait().

`add_trait_category` (category)

Adds a trait category to a class.

`add_trait_listener` (object, prefix='')

`all_trait_names` ()

Returns the list of all trait names, including implicitly defined traits.

`base_trait` (name)

Returns the base trait definition for a trait attribute.

Parameters name : str

Name of the attribute whose trait definition is returned.

`class_default_traits_view` ()

Returns the name of the default traits view for the class.

`class_editable_traits` ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current class.

`class_trait_names` (**metadata)

Returns a list of the names of all trait attributes whose definitions match the set of metadata criteria specified.

Parameters **metadata : :

Criteria for selecting trait attributes.

`class_trait_view` (name=None, view_element=None)

`class_trait_view_elements` ()

Returns the ViewElements object associated with the class.

The returned object can be used to access all the view elements associated with the class.

`class_traits` (**metadata)

Returns a dictionary containing the definitions of all of the trait attributes of the class that match the set of metadata criteria.

Parameters **metadata : :

Criteria for selecting trait attributes.

`class_visible_traits` ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the

current class, that should be GUI visible

clone_traits (*traits=None, memo=None, copy=None, **metadata*)

Clones a new object from this one, optionally copying only a specified set of traits.

Parameters **traits** : list of strings

The list of names of the trait attributes to copy.

memo : dict

A dictionary of objects that have already been copied.

copy : str

The type of copy *deep* or *shallow* to perform on any trait that does not have explicit 'copy' metadata. A value of *None* means 'copy reference'.

Returns **new** :

The newly cloned object.

configure_traits (*filename=None, view=None, kind=None, edit=True, context=None, handler=None, id='', scrollable=None, **args*)

Creates and displays a dialog box for editing values of trait attributes, as if it were a complete, self-contained GUI application.

Parameters **filename** : str

The name (including path) of a file that contains a pickled representation of the current object. When this parameter is specified, the method reads the corresponding file (if it exists) to restore the saved values of the object's traits before displaying them. If the user confirms the dialog box (by clicking **OK**), the new values are written to the file. If this parameter is not specified, the values are loaded from the in-memory object, and are not persisted when the dialog box is closed.

view : View or str

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or *None*, the **kind** attribute of the View object is used.

edit : bool

Indicates whether to display a user interface. If *filename* specifies an existing file, setting *edit* to *False* loads the saved values from that file into the object without requiring user interaction.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used

handler : Handler

A handler object used for event handling in the dialog box. If *None*, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to *True*, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns **True on success.** :

copy_traits (*other, traits=None, memo=None, copy=None, **metadata*)

Copies another object's trait attributes into this one.

Parameters **other** : object

The object whose trait attribute values should be copied.

traits : list of strings

A list of names of trait attributes to copy. If None or unspecified, the set of names returned by `trait_names()` is used. If 'all' or an empty list, the set of names returned by `all_trait_names()` is used.

memo : dict

A dictionary of objects that have already been copied.

copy : None | 'deep' | 'shallow'

The type of copy to perform on any trait that does not have explicit 'copy' metadata. A value of None means 'copy reference'.

Returns unassignable : list of strings

A list of attributes that the method was unable to copy, which is empty if all the attributes were successfully copied.

copyable_trait_names (***metadata*)

Returns the list of trait names to copy or clone by default.

default_traits_view ()

Returns the name of the default traits view for the object's class.

edit_traits (*view=None, parent=None, kind=None, context=None, handler=None, id='', scrollable=None, **args*)

Displays a user interface window for editing trait attribute values.

Parameters view : View or string

A View object (or its name) that defines a user interface for editing trait attribute values of the current object. If the view is defined as an attribute on this class, use the name of the attribute. Otherwise, use a reference to the view object. If this attribute is not specified, the View object returned by `trait_view()` is used.

parent : toolkit control

The reference to a user interface component to use as the parent window for the object's UI window.

kind : str

The type of user interface window to create. See the `traitsui.view.kind_trait` trait for values and their meanings. If *kind* is unspecified or None, the **kind** attribute of the View object is used.

context : object or dictionary

A single object or a dictionary of string/object pairs, whose trait attributes are to be edited. If not specified, the current object is used.

handler : Handler

A handler object used for event handling in the dialog box. If None, the default handler for Traits UI is used.

id : str

A unique ID for persisting preferences about this user interface, such as size and position. If not specified, no user preferences are saved.

scrollable : bool

Indicates whether the dialog box should be scrollable. When set to True, scroll bars appear on the dialog box if it is not large enough to display all of the items in the view at one time.

Returns A UI object. :

editable_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object.

get (***kwargs*)

Returns traitled class as a dict

Augments the trait get function to return a dictionary without notification handles

get_hashval (*hash_method=None*)

Return a dictionary of our items with hashes for each file.

Searches through dictionary items and if an item is a file, it calculates the md5 hash of the file contents

and stores the file name and hash value as the new key value.

However, the overall bunch hash is calculated only on the hash value of a file. The path and name of the file are not used in the overall hash calculation.

Returns `dict_withhash` : dict

Copy of our dictionary with the new file hashes included with each file.

hashvalue : str

The md5 hash value of the tra­ited spec

get_traitsfree (***kwargs*)

Returns tra­ited class as a dict

Augments the trait get function to return a dictionary without any traits. The dictionary does not contain any attributes that were Undefined

has_metadata (*name, metadata, value=None, recursive=True*)

Return has_metadata for the requested trait name in this interface

has_traits_interface (**interfaces*)

Returns whether the object implements a specified traits interface.

Parameters **interfaces* :

One or more traits Interface (sub)classes.

items ()

Name, trait generator for user modifiable traits

on_trait_change (*handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters *handler* : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If True, removes the previously-set association between *handler* and *name*; if False (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

on_trait_event (*handler, name=None, remove=False, dispatch='same', priority=False, deferred=False, target=None*)

Causes the object to invoke a handler whenever a trait attribute matching a specified pattern is modified, or removes the association.

Parameters *handler* : function

A trait notification function for the *name* trait attribute, with one of the signatures described below.

name : str

The name of the trait attribute whose value changes trigger the notification. The *name* can specify complex patterns of trait changes using an extended *name* syntax, which is described below.

remove : bool

If **True**, removes the previously-set association between *handler* and *name*; if **False** (the default), creates the association.

dispatch : str

A string indicating the thread on which notifications must be run. Possible values are:

value	dispatch
same	Run notifications on the same thread as this one.
ui	Run notifications on the UI thread. If the current thread is the UI thread, the notifications are executed immediately; otherwise, they are placed on the UI event queue.
fast	Alias for ui.
new	Run notifications in a new thread.

print_traits (*show_help=False, **metadata*)

Prints the values of all explicitly-defined, non-event trait attributes on the current object, in an easily readable format.

Parameters **show_help** : bool

Indicates whether to display additional descriptive information.

remove_trait (*name*)

Removes a trait attribute from this object.

Parameters **name** : str

Name of the attribute to remove.

Returns **result** : bool

True if the trait was successfully removed.

remove_trait_listener (*object, prefix=''*)

reset_traits (*traits=None, **metadata*)

Resets some or all of an object's trait attributes to their default values.

Parameters **traits** : list of strings

Names of trait attributes to reset.

Returns **unresetable** : list of strings

A list of attributes that the method was unable to reset, which is empty if all the attributes were successfully reset.

set (*trait_change_notify=True, **traits*)

Shortcut for setting object trait attributes.

Parameters **trait_change_notify** : bool

If **True** (the default), then each value assigned may generate a trait change notification. If **False**, then no trait change notifications will be generated. (see also: `trait_setq`)

****traits** :

Key/value pairs, the trait attributes and their values to be set

Returns **self** :

The method returns this object, after setting attributes.

set_trait_dispatch_handler (*name, klass, override=False*)

Sets a trait notification dispatch handler.

sync_trait (*trait_name, object, alias=None, mutual=True, remove=False*)

Synchronizes the value of a trait attribute on this object with a trait attribute on another object.

Parameters **name** : str

Name of the trait attribute on this object.

object : object

The object with which to synchronize.

alias : str

Name of the trait attribute on *other*; if **None** or omitted, same as *name*.

mutual : bool or int

Indicates whether synchronization is mutual (**True** or non-zero) or one-way (**False** or zero)

remove : bool or int

Indicates whether synchronization is being added (**False** or zero) or removed (**True** or

```

        non-zero)
trait (name, force=False, copy=False)
    Returns the trait definition for the name trait attribute.
    Parameters name : str
        Name of the attribute whose trait definition is to be returned.
    force : bool
        Indicates whether to return a trait definition if name is not explicitly defined.
    copy : bool
        Indicates whether to return the original trait definition or a copy.
trait_context ()
    Returns the default context to use for editing or configuring traits.
trait_get (*names, **metadata)
    Shortcut for getting object trait attributes.
    Parameters names : list of strings
        A list of trait attribute names whose values are requested.
    Returns result : dict
        A dictionary whose keys are the names passed as arguments and whose values are the
        corresponding trait values.
trait_items_event (event_trait, name, items_event)
trait_monitor (handler, remove=False)
    Adds or removes the specified handler from the list of active monitors.
    Parameters handler : function
        The function to add or remove as a monitor.
    remove : bool
        Flag indicating whether to remove (True) or add the specified handler as a monitor for
        this class.
trait_names (**metadata)
    Returns a list of the names of all trait attributes whose definitions match the set of metadata criteria
    specified.
    Parameters **metadata : :
        Criteria for selecting trait attributes.
trait_property_changed (name, old_value [, new_value ])
trait_set (trait_change_notify=True, **traits)
    Shortcut for setting object trait attributes.
    Parameters trait_change_notify : bool
        If True (the default), then each value assigned may generate a trait change notifica-
        tion. If False, then no trait change notifications will be generated. (see also: trait_setq)
    **traits : :
        Key/value pairs, the trait attributes and their values to be set
    Returns self : :
        The method returns this object, after setting attributes.
trait_setq (**traits)
    Shortcut for setting object trait attributes.
    Parameters **traits : :
        Key/value pairs, the trait attributes and their values to be set. No trait change notifica-
        tions will be generated for any values assigned (see also: trait_set).
    Returns self : :
        The method returns this object, after setting attributes.
trait_subclasses (all=False)
    Returns a list of the immediate (or all) subclasses of this class.
    Parameters all : bool
        Indicates whether to return all subclasses of this class. If False, only immediate sub-
        classes are returned.
trait_view (name=None, view_element=None)

```

Gets or sets a ViewElement associated with an object's class.

Parameters **name** : str

Name of a view element

view_element : ViewElement

View element to associate

Returns A view element. :

trait_view_elements ()

Returns the ViewElements object associated with the object's class.

The returned object can be used to access all the view elements associated with the class.

trait_views (klass=None)

Returns a list of the names of all view elements associated with the current object's class.

Parameters **klass** : class

A class, such that all returned names must correspond to instances of this class. Possible values include:

Group

Item

View

ViewElement

ViewSubElement

traits (**metadata)

Returns a dictionary containing the definitions of all of the trait attributes of this object that match the set of *metadata* criteria.

Parameters ****metadata** : :

Criteria for selecting trait attributes.

traits_init ()

traits_initiated ([True])

validate_trait (name, value)

Validates whether a value is legal for a trait.

Returns the validated value if it is valid.

visible_traits ()

Returns an alphabetically sorted list of the names of non-event trait attributes associated with the current object, that should be GUI visible

wrappers = {'ui': <class 'traits.trait_notifiers.FastUITraitChangeNotifyWrapper'>, 'new': <class 'traits.trait_notif

Function

`nipytype.interfaces.matlab.get_matlab_command()`

pipeline.engine.base

Module: `pipeline.engine.base`

Inheritance diagram for `nipytype.pipeline.engine.base`:



base.EngineBase

faces

Defines functionality for pipelined execution of inter-

The *EngineBase* class implements the more general view of a task.

EngineBase

class nipytype.pipeline.engine.base.**EngineBase** (*name=None, base_dir=None*)

Bases: object

Defines common attributes and functions for workflows and nodes.

Methods

<i>clone</i> (name)	Clone an EngineBase object
<i>load</i> (filename)	
<i>save</i> ([filename])	

__init__ (*name=None, base_dir=None*)

Initialize base parameters of a workflow or node

Parameters **name** : string (mandatory)

Name of this node. Name must be alphanumeric and not contain any special characters (e.g., '.', '@').

base_dir : string

base output directory (will be hashed before creations) default=None, which results in the use of mkdtemp

clone (*name*)

Clone an EngineBase object

Parameters **name** : string (mandatory)

A clone of node or workflow must have a new name

fullname

inputs

itername

load (*filename*)

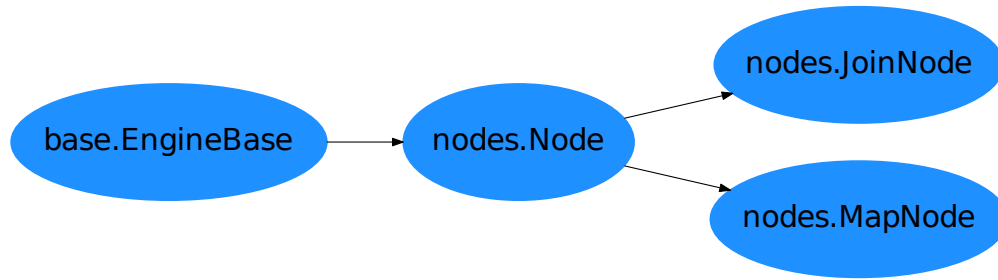
outputs

save (*filename=None*)

pipeline.engine.nodes

Module: pipeline.engine.nodes

Inheritance diagram for nipytype.pipeline.engine.nodes:



Defines functionality for pipelined execution of interfaces
 The *Node* class provides core functionality for batch processing.

Classes

JoinNode

class `nipyte.pipeline.engine.nodes.JoinNode`(*interface, name, joinsource, joinfield=None, unique=False, **kwargs*)

Bases: `nipyte.pipeline.engine.nodes.Node`

Wraps interface objects that join inputs into a list.

Examples

```

>>> import nipyte.pipeline.engine as pe
>>> from nipyte import Node, JoinNode, Workflow
>>> from nipyte.interfaces.utility import IdentityInterface
>>> from nipyte.interfaces import (ants, dcm2nii, fsl)
>>> wf = Workflow(name='preprocess')
>>> inputspec = Node(IdentityInterface(fields=['image']),
...                  name='inputspec')
>>> inputspec.iterables = [('image',
...                        ['img1.nii', 'img2.nii', 'img3.nii'])]
>>> img2flt = Node(fsl.ImageMaths(out_data_type='float'),
...               name='img2flt')
>>> wf.connect(inputspec, 'image', img2flt, 'in_file')
>>> average = JoinNode(ants.AverageImages(), joinsource='inputspec',
...                   joinfield='images', name='average')
>>> wf.connect(img2flt, 'out_file', average, 'images')
>>> realign = Node(fsl.FLIRT(), name='realign')
>>> wf.connect(img2flt, 'out_file', realign, 'in_file')
>>> wf.connect(average, 'output_average_image', realign, 'reference')
>>> strip = Node(fsl.BET(), name='strip')
>>> wf.connect(realign, 'out_file', strip, 'in_file')
  
```

Methods

`clone(name)`

Clone an EngineBase object

Continued on next page

Table 3.38 – continued from previous page

<code>get_output(parameter)</code>	Retrieve a particular output of the node
<code>hash_exists([updatehash])</code>	
<code>help()</code>	Print interface help
<code>load(filename)</code>	
<code>output_dir()</code>	Return the location of the output directory for the node
<code>run([updatehash])</code>	Execute the node in its directory.
<code>save([filename])</code>	
<code>set_input(parameter, val)</code>	Set interface input value
<code>update(**opts)</code>	
<code>write_report([report_type, cwd])</code>	

`__init__` (*interface, name, joinsource, joinfield=None, unique=False, **kwargs*)

Parameters **interface** : interface object

node specific interface (fsl.Bet(), spm.Coregister())

name : alphanumeric string

node specific name

joinsource : node name

name of the join predecessor iterable node

joinfield : string or list of strings

name(s) of list input fields that will be aggregated. The default is all of the join node input fields.

unique : flag indicating whether to ignore duplicate input values

See Node docstring for additional keyword arguments. :

clone (*name*)

Clone an EngineBase object

Parameters **name** : string (mandatory)

A clone of node or workflow must have a new name

fullname

get_output (*parameter*)

Retrieve a particular output of the node

hash_exists (*updatehash=False*)

help ()

Print interface help

inputs

The JoinNode inputs include the join field overrides.

interface

Return the underlying interface object

itername

joinsource

load (*filename*)

output_dir ()

Return the location of the output directory for the node

outputs

Return the output fields of the underlying interface

result

run (*updatehash=False*)

Execute the node in its directory.

Parameters **updatehash: boolean** :

Update the hash stored in the output directory

save (*filename=None*)

set_input (*parameter, val*)

Set interface input value

```
update (**opts)
write_report (report_type=None, cwd=None)
```

MapNode

class nipyype.pipeline.engine.nodes.**MapNode**(*interface*, *iterfield*, *name*, *serial=False*, *nested=False*, ***kwargs*)

Bases: *nipyype.pipeline.engine.nodes.Node*

Wraps interface objects that need to be iterated on a list of inputs.

Examples

```
>>> from nipyype import MapNode
>>> from nipyype.interfaces import fsl
>>> realign = MapNode(fsl.MCFLIRT(), 'in_file', 'realign')
>>> realign.inputs.in_file = ['functional.nii',
...                           'functional2.nii',
...                           'functional3.nii']
>>> realign.run()
```

Methods

<i>clone</i> (name)	Clone an EngineBase object
<i>get_output</i> (parameter)	Retrieve a particular output of the node
<i>get_subnodes</i> ()	
<i>hash_exists</i> ([updatehash])	
<i>help</i> ()	Print interface help
<i>load</i> (filename)	
<i>num_subnodes</i> ()	
<i>output_dir</i> ()	Return the location of the output directory for the node
<i>run</i> ([updatehash])	Execute the node in its directory.
<i>save</i> ([filename])	
<i>set_input</i> (parameter, val)	Set interface input value or nodewrapper attribute
<i>update</i> (**opts)	
<i>write_report</i> ([report_type, cwd])	

__init__ (*interface*, *iterfield*, *name*, *serial=False*, *nested=False*, ***kwargs*)

Parameters **interface** : interface object

node specific interface (fsl.Bet(), spm.Coregister())

iterfield : string or list of strings

name(s) of input fields that will receive a list of whatever kind of input they take. the node will be run separately for each value in these lists. for more than one input, the values are paired (i.e. it does not compute a combinatorial product).

name : alphanumeric string

node specific name

serial : boolean

flag to enforce executing the jobs of the mapnode in a serial manner rather than parallel

nested : boolean

support for nested lists, if set the input list will be flattened before running, and the nested list structure of the outputs will be resored

See Node docstring for additional keyword arguments. :

```

clone (name)
    Clone an EngineBase object
    Parameters name : string (mandatory)
        A clone of node or workflow must have a new name
fullname
get_output (parameter)
    Retrieve a particular output of the node
get_subnodes ()
hash_exists (updatehash=False)
help ()
    Print interface help
inputs
interface
    Return the underlying interface object
itername
load (filename)
num_subnodes ()
output_dir ()
    Return the location of the output directory for the node
outputs
result
run (updatehash=False)
    Execute the node in its directory.
    Parameters updatehash: boolean :
        Update the hash stored in the output directory
save (filename=None)
set_input (parameter, val)
    Set interface input value or nodewrapper attribute
    Priority goes to interface.
update (**opts)
write_report (report_type=None, cwd=None)

```

Node

```

class nipyype.pipeline.engine.nodes.Node (interface, name, iterables=None, iter-
                                         source=None, synchronize=False, over-
                                         write=None, needed_outputs=None,
                                         run_without_submitting=False, n_procs=1,
                                         mem_gb=None, **kwargs)

```

Bases: `nipyype.pipeline.engine.base.EngineBase`

Wraps interface objects for use in pipeline

A Node creates a sandbox-like directory for executing the underlying interface. It will copy or link inputs into this directory to ensure that input data are not overwritten. A hash of the input state is used to determine if the Node inputs have changed and whether the node needs to be re-executed.

Examples

```

>>> from nipyype import Node
>>> from nipyype.interfaces import spm
>>> realign = Node(spm.Realign(), 'realign')
>>> realign.inputs.in_files = 'functional.nii'
>>> realign.inputs.register_to_mean = True
>>> realign.run()

```

Methods

<code>clone(name)</code>	Clone an EngineBase object
<code>get_output(parameter)</code>	Retrieve a particular output of the node
<code>hash_exists([updatehash])</code>	
<code>help()</code>	Print interface help
<code>load(filename)</code>	
<code>output_dir()</code>	Return the location of the output directory for the node
<code>run([updatehash])</code>	Execute the node in its directory.
<code>save([filename])</code>	
<code>set_input(parameter, val)</code>	Set interface input value
<code>update(**opts)</code>	
<code>write_report([report_type, cwd])</code>	

__init__(*interface*, *name*, *iterables=None*, *itersource=None*, *synchronize=False*, *overwrite=None*, *needed_outputs=None*, *run_without_submitting=False*, *n_procs=1*, *mem_gb=None*, ***kwargs*)

Parameters **interface** : interface object

node specific interface (fsl.Bet(), spm.Coregister())

name : alphanumeric string

node specific name

iterables : generator

Input field and list to iterate using the pipeline engine for example to iterate over different frac values in fsl.Bet() for a single field the input can be a tuple, otherwise a list of tuples

```
node.iterables = ('frac', [0.5, 0.6, 0.7])
node.iterables = [('fwhm', [2, 4]), ('fieldx', [0.5, 0.6, 0.7])]
```

If this node has an itersource, then the iterables values is a dictionary which maps an iterable source field value to the target iterables field values, e.g.:

```
inputspec.iterables = ('images', ['img1.nii', 'img2.nii'])
node.itersource = ('inputspec', ['frac'])
node.iterables = ('frac', {'img1.nii': [0.5, 0.6],
                           'img2.nii': [0.6, 0.7]})
```

If this node's synchronize flag is set, then an alternate form of the iterables is a [fields, values] list, where fields is the list of iterated fields and values is the list of value tuples for the given fields, e.g.:

```
node.synchronize = True
node.iterables = [('frac', 'threshold'),
                  [(0.5, True),
                   (0.6, False)]]
```

itersource: tuple :

The (name, fields) iterables source which specifies the name of the predecessor iterable node and the input fields to use from that source node. The output field values comprise the key to the iterables parameter value mapping dictionary.

synchronize: boolean :

Flag indicating whether iterables are synchronized. If the iterables are synchronized, then this iterable node is expanded once per iteration over all of the iterables values. Otherwise, this iterable node is expanded once per each permutation of the iterables values.

overwrite : Boolean
Whether to overwrite contents of output directory if it already exists. If directory exists and hash matches it assumes that process has been executed

needed_outputs : list of output_names
Force the node to keep only specific outputs. By default all outputs are kept. Setting this attribute will delete any output files and directories from the node's working directory that are not part of the *needed_outputs*.

run_without_submitting : boolean
Run the node without submitting to a job engine or to a multiprocessing pool

clone (*name*)
Clone an EngineBase object
Parameters **name** : string (mandatory)
A clone of node or workflow must have a new name

fullname

get_output (*parameter*)
Retrieve a particular output of the node

hash_exists (*updatehash=False*)

help ()
Print interface help

inputs
Return the inputs of the underlying interface

interface
Return the underlying interface object

itername

load (*filename*)

output_dir ()
Return the location of the output directory for the node

outputs
Return the output fields of the underlying interface

result

run (*updatehash=False*)
Execute the node in its directory.
Parameters **updatehash**: boolean :
Update the hash stored in the output directory

save (*filename=None*)

set_input (*parameter, val*)
Set interface input value

update (***opts*)

write_report (*report_type=None, cwd=None*)

pipeline.engine.utils

Module: pipeline.engine.utils

Utility routines for workflow graphs

Functions

`nipyte.pipeline.engine.utils.clean_working_directory` (*outputs, cwd, inputs, needed_outputs, config, files2keep=None, dirs2keep=None*)

Removes all files not needed for further analysis from the directory

```
nipyype.pipeline.engine.utils.count_iterables(iterables, synchronize=False)
```

Return the number of iterable expansion nodes.

If synchronize is True, then the count is the maximum number of iterables value lists. Otherwise, the count is the product of the iterables value list sizes.

```
nipyype.pipeline.engine.utils.evaluate_connect_function(function_source, args,
                                                         first_arg)
```

```
nipyype.pipeline.engine.utils.expand_iterables(iterables, synchronize=False)
```

```
nipyype.pipeline.engine.utils.export_graph(graph_in, base_dir=None,
                                              show=False, use_execgraph=False,
                                              show_connectioninfo=False, dotfile-
                                              name='graph.dot', format='png',
                                              simple_form=True)
```

Displays the graph layout of the pipeline

This function requires that pygraphviz and matplotlib are available on the system.

Parameters `show` : boolean

Indicate whether to generate pygraphviz output fromn :

`networkx`. default [False] :

`use_execgraph` : boolean

Indicates whether to use the specification graph or the :

`execution graph`. default [False] :

`show_connectioninfo` : boolean

Indicates whether to show the edge data on the graph. This :

makes the graph rather cluttered. default [False] :

```
nipyype.pipeline.engine.utils.format_dot(dotfilename, format='png')
```

Dump a directed graph (Linux only; install via *brew* on OSX)

```
nipyype.pipeline.engine.utils.format_node(node, format='python', include-
                                          config=False)
```

Format a node in a given output syntax.

```
nipyype.pipeline.engine.utils.generate_expanded_graph(graph_in)
```

Generates an expanded graph based on node parameterization

Parameterization is controlled using the *iterables* field of the pipeline elements. Thus if there are two nodes with iterables `a=[1,2]` and `b=[3,4]` this procedure will generate a graph with sub-graphs parameterized as `(a=1,b=3)`, `(a=1,b=4)`, `(a=2,b=3)` and `(a=2,b=4)`.

```
nipyype.pipeline.engine.utils.get_all_files(infile)
```

```
nipyype.pipeline.engine.utils.get_levels(G)
```

```
nipyype.pipeline.engine.utils.get_print_name(node, simple_form=True)
```

Get the name of the node

For example, a node containing an instance of `interfaces.fsl.BET` would be called `nodename.BET.fsl`

```
nipyype.pipeline.engine.utils.make_output_dir(outdir)
```

Make the output_dir if it doesn't exist.

Parameters `outdir` : output directory to create

```
nipyype.pipeline.engine.utils.merge_bundles(g1, g2)
```

```
nipyype.pipeline.engine.utils.merge_dict(d1, d2, merge=<function <lambda>>)
```

Merges two dictionaries, non-destructively, combining values on duplicate keys as defined by the optional merge function. The default behavior replaces the values in `d1` with corresponding values in `d2`. (There is no other generally applicable merge strategy, but often you'll have homogeneous types in your dicts, so specifying a merge technique can be valuable.)

Examples:

```
>>> d1 = {'a': 1, 'c': 3, 'b': 2}
>>> d2 = merge_dict(d1, d1)
>>> len(d2)
3
>>> [d2[k] for k in ['a', 'b', 'c']]
[1, 2, 3]
```



```
>>> d3 = merge_dict(d1, d1, lambda x,y: x+y)
>>> len(d3)
3
>>> [d3[k] for k in ['a', 'b', 'c']]
[2, 4, 6]
```

`nipyype.pipeline.engine.utils.modify_paths(object, relative=True, basedir=None)`

Convert paths in data structure to either full paths or relative paths

Supports combinations of lists, dicts, tuples, str

Parameters `relative` : boolean indicating whether paths should be set relative to the current directory

`basedir` : default `os.getcwd()`

what base directory to use as default

`nipyype.pipeline.engine.utils.synchronize_iterables(iterables)`

Synchronize the given iterables in item-wise order.

Return: the {field: value} dictionary list

Examples

```
>>> from nipyype.pipeline.engine.utils import synchronize_iterables
>>> iterables = dict(a=lambda: [1, 2], b=lambda: [3, 4])
>>> synced = synchronize_iterables(iterables)
>>> synced == [{'a': 1, 'b': 3}, {'a': 2, 'b': 4}]
True
>>> iterables = dict(a=lambda: [1, 2], b=lambda: [3], c=lambda: [4, 5, 6])
>>> synced = synchronize_iterables(iterables)
>>> synced == [{'a': 1, 'b': 3, 'c': 4}, {'a': 2, 'c': 5}, {'c': 6}]
True
```

`nipyype.pipeline.engine.utils.topological_sort(graph, depth_first=False)`

Returns a depth first sorted order if `depth_first` is True

`nipyype.pipeline.engine.utils.walk(children, level=0, path=None, username=True)`

Generate all the full paths in a tree, as a dict.

Examples

```
>>> from nipyype.pipeline.engine.utils import walk
>>> iterables = [('a', lambda: [1, 2]), ('b', lambda: [3, 4])]
>>> [val['a'] for val in walk(iterables)]
[1, 1, 2, 2]
>>> [val['b'] for val in walk(iterables)]
[3, 4, 3, 4]
```

`nipyype.pipeline.engine.utils.walk_files(cwd)`

`nipyype.pipeline.engine.utils.walk_outputs(object)`

Extract every file and directory from a python structure

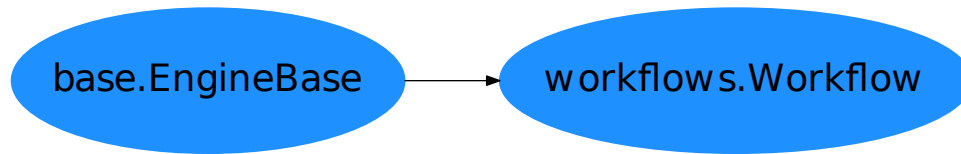
`nipyype.pipeline.engine.utils.write_workflow_prov(graph, filename=None, format='all')`

Write W3C PROV Model JSON file

pipeline.engine.workflows

Module: pipeline.engine.workflows

Inheritance diagram for `nipyype.pipeline.engine.workflows`:



Defines functionality for pipelined execution of interfaces
The *Workflow* class provides core functionality for batch processing.

Workflow

class nipytype.pipeline.engine.workflows.**Workflow** (*name*, *base_dir*=None)

Bases: *nipytype.pipeline.engine.base.EngineBase*

Controls the setup and execution of a pipeline of processes.

Methods

<i>add_nodes</i> (nodes)	Add nodes to a workflow
<i>clone</i> (name)	Clone a workflow
<i>connect</i> (*args, **kwargs)	Connect nodes in the pipeline.
<i>disconnect</i> (*args)	Disconnect nodes See the docstring for connect for format.
<i>export</i> ([filename, prefix, format, ...])	Export object into a different format
<i>get_node</i> (name)	Return an internal node by name
<i>list_node_names</i> ()	List names of all nodes in a workflow
<i>load</i> (filename)	
<i>remove_nodes</i> (nodes)	Remove nodes from a workflow
<i>run</i> ([plugin, plugin_args, updatehash])	Execute the workflow
<i>save</i> ([filename])	
<i>write_graph</i> ([dotfilename, graph2use, ...])	Generates a graphviz dot file and a png file
<i>write_hierarchical_dotfile</i> ([dotfilename, ...])	

__init__ (*name*, *base_dir*=None)

Create a workflow object.

Parameters **name** : alphanumeric string
unique identifier for the workflow

base_dir : string, optional
path to workflow storage

add_nodes (*nodes*)

Add nodes to a workflow

Parameters **nodes** : list
A list of EngineBase-based objects

clone (*name*)

Clone a workflow

Note: Will reset attributes used for executing workflow. See `_init_runtime_fields`.

Parameters name: alphanumeric name :

unique name for the workflow

connect (*args, **kwargs)

Connect nodes in the pipeline.

This routine also checks if inputs and outputs are actually provided by the nodes that are being connected.

Creates edges in the directed graph using the nodes and edges specified in the *connection_list*. Uses the NetworkX method `DiGraph.add_edges_from`.

Parameters args : list or a set of four positional arguments

Four positional arguments of the form:

```
connect(source, sourceoutput, dest, destinput)
```

source : nodewrapper node sourceoutput : string (must be in source.outputs) dest :
nodewrapper node destinput : string (must be in dest.inputs)

A list of 3-tuples of the following form:

```
[(source, target,
  [('sourceoutput/attribute', 'targetinput'),
   ...]),
 ...]
```

Or:

```
[(source, target, [(('sourceoutput1', func, arg2, ...),
                    'targetinput'), ...]),
 ...]
sourceoutput1 will always be the first argument to func
and func will be evaluated and the results sent ot targetinput

currently func needs to define all its needed imports within
↳the
function as we use the inspect module to get at the source
↳code
and execute it remotely
```

disconnect (*args)

Disconnect nodes See the docstring for connect for format.

export (filename=None, prefix='output', format='python', include_config=False)

Export object into a different format

Parameters filename: string :

file to save the code to; overrides prefix

prefix: string :

prefix to use for output file

format: string :

one of "python"

include_config: boolean :

whether to include node and workflow config values

fullname

get_node (name)

Return an internal node by name

inputs

itername

list_node_names ()

List names of all nodes in a workflow

load (*filename*)
outputs
remove_nodes (*nodes*)
Remove nodes from a workflow
Parameters nodes : list
A list of EngineBase-based objects
run (*plugin=None, plugin_args=None, updatehash=False*)
Execute the workflow
Parameters plugin: plugin name or object :
Plugin to use for execution. You can create your own plugins for execution.
plugin_args : dictionary containing arguments to be sent to plugin constructor. see individual plugin doc strings for details.
save (*filename=None*)
write_graph (*dotfilename='graph.dot', graph2use='hierarchical', format='png', simple_form=True*)
Generates a graphviz dot file and a png file
Parameters graph2use: 'orig', 'hierarchical' (default), 'flat', 'exec', 'colored' :
orig - creates a top level graph without expanding internal workflow nodes; flat - expands workflow nodes recursively; hierarchical - expands workflow nodes recursively with a notion on hierarchy; colored - expands workflow nodes recursively with a notion on hierarchy in color; exec - expands workflows to depict iterables
format: 'png', 'svg' :
simple_form: boolean (default: True) :
Determines if the node name used in the graph should be of the form 'nodename (package)' when True or 'nodename.Class.package' when False.
write_hierarchical_dotfile (*dotfilename=None, colored=False, simple_form=True*)

sphinxext.plot_workflow

Module: sphinxext.plot_workflow

Inheritance diagram for `nipytype.sphinxext.plot_workflow`:



plot_workflow.ImageFile



plot_workflow.GraphError

nipy.sphinxext.plot_workflow – Workflow plotting extension

A directive for including a nipy workflow graph in a Sphinx document.

This code is forked from the plot_figure sphinx extension of matplotlib.

By default, in HTML output, *workflow* will include a .png file with a link to a high-res .png. In LaTeX output, it will include a .pdf. The source code for the workflow may be included as **inline content** to the directive *workflow*:

```
.. workflow ::
    :graph2use: flat
    :simple_form: no

    from nipy.workflows.dmri.camino.connectivity_mapping import create_
    ↪connectivity_pipeline
    wf = create_connectivity_pipeline()
```

For example, the following graph has been generated inserting the previous code block in this documentation:

Options

The **workflow** directive supports the following options:

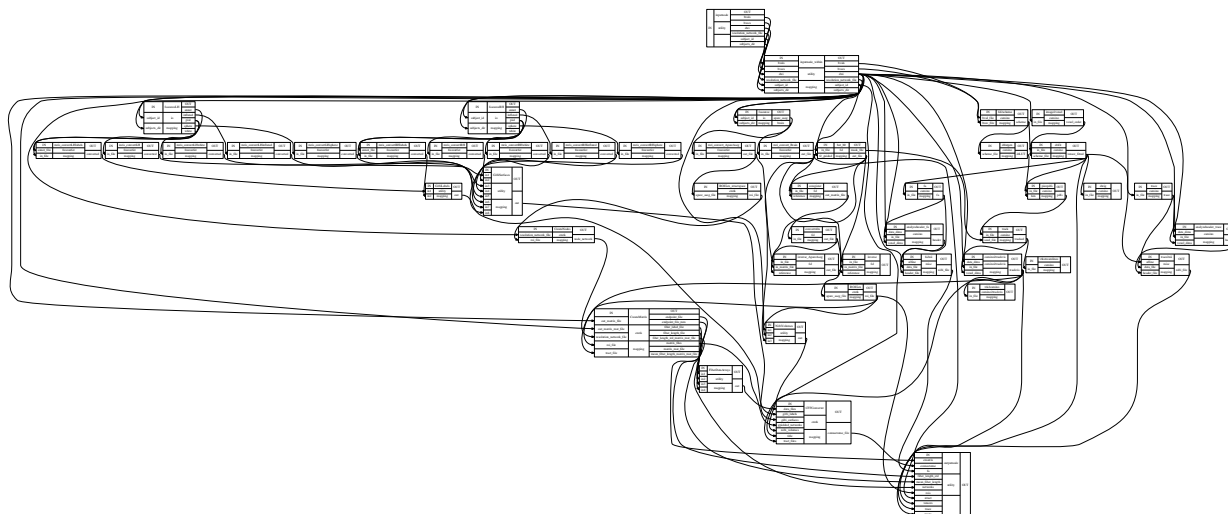
graph2use [{‘hierarchical’, ‘colored’, ‘flat’, ‘orig’, ‘exec’}] Specify the type of graph to be generated.

simple_form: bool Whether the graph will be in detailed or simple form.

format [{‘python’, ‘doctest’}] Specify the format of the input

include-source [bool] Whether to display the source code. The default can be changed using the *workflow_include_source* variable in conf.py

encoding [str] If this source file is in a non-UTF8 or non-ASCII encoding, the encoding must be specified using the *:encoding:* option. The encoding will not be inferred using the `-- coding --`



metacomment.

Additionally, this directive supports all of the options of the *image* directive, except for *target* (since workflow will add its own target). These include *alt*, *height*, *width*, *scale*, *align* and *class*.

Configuration options

The workflow directive has the following configuration options:

graph2use Select a graph type to use

simple_form determines if the node name shown in the visualization is either of the form `nodename (package)` when set to `True` or `nodename.Class.package` when set to `False`.

wf_include_source Default value for the include-source option

wf html show source link Whether to show a link to the source in HTML.

wf_pre_code Code that should be executed before each workflow.

wf_basedir Base directory, to which `workflow::` file names are relative to. (If `None` or empty, file names are relative to the directory where the file containing the directive is.)

wf formats

File formats to generate. List of tuples or strings:: [(suffix, dpi), suffix, ...]

that determine the file format and the DPI. For entries whose DPI was omitted, sensible defaults are chosen. When passing from the command line through `sphinx_build` the list should be passed as `suffix:dpi,suffix:dpi, ...`

wf html show formats Whether to show links to the files in HTML.

wf_rcparams A dictionary containing any non-standard rcParams that should be applied before each workflow.

wf_apply_rcparams By default, rcParams are applied when *context* option is not used in a workflow directive. This configuration option overrides this behavior and applies rcParams before each workflow.

wf_working_directory By default, the working directory will be changed to the directory of the example, so the code can get at its data files, if any. Also its path will be added to *sys.path* so it can import any helper modules sitting beside it. This configuration option can be used to specify a central directory (also added to *sys.path*) where data files and helper modules for all code are located.

wf_template Provide a customized template for preparing restructured text.

Classes

GraphError

class nipyne.sphinxext.plot_workflow.**GraphError**

Bases: RuntimeError

Methods

<i>with_traceback</i>	Exception.with_traceback(tb)	–	set
	self.__traceback__ to tb and return self.		

__init__ ()

Initialize self. See help(type(self)) for accurate signature.

args

with_traceback ()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

ImageFile

class nipyne.sphinxext.plot_workflow.**ImageFile** (*basename, dirname*)

Bases: object

Methods

<i>filename</i> (fmt)
<i>filenames</i> ()

__init__ (*basename, dirname*)

filename (*fmt*)

filenames ()

Functions

nipyne.sphinxext.plot_workflow.**contains_doctest** (*text*)

nipyne.sphinxext.plot_workflow.**get_wf_formats** (*config*)

nipyne.sphinxext.plot_workflow.**mark_wf_labels** (*app, document*)

To make graphs referenceable, we need to move the reference from the “htmlonly” (or “latexonly”) node to the actual figure node itself.

nipyne.sphinxext.plot_workflow.**out_of_date** (*original, derived*)

Returns True if derivative is out-of-date wrt original, both of which are full file paths.

nipyne.sphinxext.plot_workflow.**remove_coding** (*text*)

Remove the coding comment, which exec doesn’t like.

nipyne.sphinxext.plot_workflow.**render_figures** (*code, code_path, output_dir,*
output_base, context, func-
tion_name, config, graph2use,
simple_form, context_reset=False,
close_figs=False)

Run a nipyne workflow creation script and save the graph in *output_dir*. Save the images under *output_dir* with file names derived from *output_base*

nipyne.sphinxext.plot_workflow.**run** (*arguments, content, options, state_machine, state,*
lineno)

nipyne.sphinxext.plot_workflow.**run_code** (*code, code_path, ns=None, func-*
tion_name=None)

Import a Python module from a path, and run the function given by name, if function_name is not None.

```
nipytype.sphinxext.plot_workflow.setup(app)
```

```
nipytype.sphinxext.plot_workflow.unescape_doctest(text)
```

Extract code from a piece of text, which contains either Python code or doctests.

```
nipytype.sphinxext.plot_workflow.wf_directive(name, arguments, options, content,  
                                                lineno, content_offset, block_text, state,  
                                                state_machine)
```


Release 0.13.1

Date May 21, 2017, 03:41 PDT

Since nipy is part of the [NIPY](#) project, we follow the same conventions documented in the [NIPY Developers Guide](#). For bleeding-edge version help see [Nightly documentation](#)

Interface Specifications

Before you start

Nipype is a young project maintained by an enthusiastic group of developers. Even though the documentation might be sparse or cryptic at times we strongly encourage you to contact us on the official nipype developers mailing list in case of any troubles: nipy-devel@neuroimaging.scipy.org (we are sharing a mailing list with the nipy community therefore please add [nipype] to the message title).

Overview

We're using the [Enthought Traits](#) package for all of our inputs and outputs. Traits allows us to validate user inputs and provides a mechanism to handle all the *special cases* in a simple and concise way through metadata. With the metadata, each input/output can have an optional set of metadata attributes (described in more detail below). The machinery for handling the metadata is located in the base classes, so all subclasses use the same code to handle these cases. This is in contrast to our previous code where every class defined its own `_parse_inputs`, `run` and `aggregate_outputs` methods to handle these cases. Which of course leads to a dozen different ways to solve the same problem.

Traits is a big package with a lot to learn in order to take full advantage of. But don't be intimidated! To write a Nipype Trait Specification, you only need to learn a few of the basics of Traits. Here are a few starting points in the documentation:

- What are Traits? The [Introduction in the User Manual](#) gives a brief description of the functionality traits provides.
- Traits and metadata. The [second section of the User Manual](#) gives more details on traits and how to use them. Plus there a section describing metadata, including the metadata all traits have.
- If your interested in more of a *big picture* overview, [Gael wrote a good tutorial](#) that shows how to write a scientific application using traits for the benefit of the generated UI components. (For now, Nipype is not taking advantage of the generated UI feature of traits.)

Traits version

We're using Traits version 3.x which can be install as part of [EPD](#) or from [pypi](#)

More documentation

Not everything is documented in the User Manual, in those cases the [enthought-dev mailing list](#) or the [API docs](#) is your next place to look.

Nipyre Interface Specifications

Each interface class defines two specifications: 1) an InputSpec and 2) an OutputSpec. Each of these are prefixed with the class name of the interfaces. For example, Bet has these specs:

- BETInputSpec
- BETOutputSpec

Each of these Specs are classes, derived from a base `TraitedSpec` class (more on these below). The `InputSpec` consists of attributes which correspond to different parameters for the tool they wrap/interface. In the case of a command-line tool like Bet, the `InputSpec` attributes correspond to the different command-line parameters that can be passed to Bet. If you are familiar with the Nipyre 0.2 code-base, these attributes are the same as the keys in the `opt_map` dictionaries. When an interfaces class is instantiated, the `InputSpec` is bound to the `inputs` attribute of that object. Below is an example of how the `inputs` appear to a user for Bet:

```
>>> from nipyre.interfaces import fsl
>>> bet = fsl.BET()
>>> type(bet.inputs)
<class 'nipyre.interfaces.fsl.preprocess.BETInputSpec'>
>>> bet.inputs.<TAB>
bet.inputs.__class__          bet.inputs.center
bet.inputs.__delattr__       bet.inputs.environ
bet.inputs.__doc__           bet.inputs.frac
bet.inputs.__getattr__       bet.inputs.functional
bet.inputs.__hash__          bet.inputs.hashval
bet.inputs.__init__          bet.inputs.infile
bet.inputs.__new__           bet.inputs.items
bet.inputs.__reduce__        bet.inputs.mask
bet.inputs.__reduce_ex__     bet.inputs.mesh
bet.inputs.__repr__          bet.inputs.nooutput
bet.inputs.__setattr__       bet.inputs.outfile
bet.inputs.__str__           bet.inputs.outline
bet.inputs._generate_handlers bet.inputs.outputtype
bet.inputs._get_hashval      bet.inputs.radius
bet.inputs._hash_infile      bet.inputs.reduce_bias
bet.inputs._xor_inputs       bet.inputs.skull
bet.inputs._xor_warn         bet.inputs.threshold
bet.inputs.args              bet.inputs.vertical_gradient
```

Each Spec inherits from a parent Spec. The parent Specs provide attribute(s) that are common to all child classes. For example, FSL InputSpecs inherit from `interfaces.fsl.base.FSLTraitedSpec`. `FSLTraitedSpec` defines an `outputtype` attribute, which stores the file type (NIFTI, NIFTI_PAIR, etc...) for all generated output files.

InputSpec class hierarchy

Below is the current class hierarchy for `InputSpec` classes (from base class down to subclasses):

`TraitedSpec`: Nipyre's primary base class for all Specs. Provides initialization, some nipyre-specific methods and any trait handlers we define. Inherits from `traits.HasTraits`.

`BaseInterfaceInputSpec`: Defines inputs common to all Interfaces (`ignore_exception`). If in doubt inherit from this.

CommandLineInputSpec: Defines inputs common to all command-line classes (args and environ)
 FSLTraitedSpec: Defines inputs common to all FSL classes (outputtype)
 SPMCommandInputSpec: Defines inputs common to all SPM classes (matlab_cmd, path, and mfile)
 FSTraitedSpec: Defines inputs common to all FreeSurfer classes (subjects_dir)
 MatlabInputSpec: Defines inputs common to all Matlab classes (script, nodesktop, nosplash, logfile, single_comp_thread, mfile, script_file, and paths)
 SlicerCommandLineInputSpec: Defines inputs common to all Slicer classes (module)

Most developers will only need to code at the the interface-level (i.e. implementing custom class inheriting from one of the above classes).

Output Specs

The OutputSpec defines the outputs that are generated, or possibly generated depending on inputs, by the tool. OutputSpecs inherit from `interfaces.base.TraitedSpec` directly.

Traited Attributes

Each specification attribute is an instance of a Trait class. These classes encapsulate many standard Python types like Float and Int, but with additional behavior like type checking. (*See the documentation on traits for more information on these trait types.*) To handle unique behaviors of our attributes we use traits metadata. These are keyword arguments supplied in the initialization of the attributes. The base classes `BaseInterface` and `CommandLine` (defined in `nipy.interfaces.base`) check for the existence/or value of these metadata and handle the inputs/outputs accordingly. For example, all mandatory parameters will have the `mandatory = True` metadata:

```
class BetInputSpec(FSLTraitedSpec):
    infile = File(exists=True,
                  desc='input file to skull strip',
                  argstr='%s', position=0, mandatory=True)
```

Common

exists For files, use `nipy.interfaces.base.File` as the trait type. If the file must exist for the tool to execute, specify `exists = True` in the initialization of `File` (as shown in `BetInputSpec` above). This will trigger the underlying traits code to confirm the file assigned to that *input* actually exists. If it does not exist, the user will be presented with an error message:

```
>>> bet.inputs.infile = 'does_not_exist.nii'
-----
Traceback (most recent call last):
  File "<ipython console>", line 1, in <module>
  File "/Users/cburns/local/lib/python2.5/site-packages/nipy/interfaces/
↳base.py", line 76, in validate
    self.error( object, name, value )
  File "/Users/cburns/local/lib/python2.5/site-packages/enthought/traits/
↳trait_handlers.py", line 175, in error
    value )
TraitError: The 'infile' trait of a BetInputSpec instance must be a file
name, but a value of 'does_not_exist.nii' <type 'str'> was specified.
```

hash_files To be used with inputs that are defining output filenames. When this flag is set to false any Nipyre will not try to hash any files described by this input. This is useful to avoid rerunning when the specified output file already exists and has changed.

desc All trait objects have a set of default metadata attributes. `desc` is one of those and is used as a simple, one-line docstring. The `desc` is printed when users use the `help()` methods.

Required: This metadata is required by all nipyre interface classes.

usedefault Set this metadata to True when the *default value* for the trait type of this attribute is an acceptable value. All trait objects have a default value, `traits.Int` has a default of 0, `traits.Float` has a default of 0.0, etc... You can also define a default value when you define the class. For example, in the code below all objects of `Foo` will have a default value of 12 for `x`:

```
>>> import enthought.traits.api as traits
>>> class Foo(traits.HasTraits):
...     x = traits.Int(12)
...     y = traits.Int
...
>>> foo = Foo()
>>> foo.x
12
>>> foo.y
0
```

Nipyre only passes inputs on to the underlying package if they have been defined (more on this later). So if you specify `usedefault = True`, you are telling the parser to pass the default value on to the underlying package. Let's look at the `InputSpec` for SPM Realign:

```
class RealignInputSpec(BaseInterfaceInputSpec):
    jobtype = traits.Enum('estwrite', 'estimate', 'write',
                          desc='one of: estimate, write, estwrite',
                          usedefault=True)
```

Here we've defined `jobtype` to be an enumerated trait type, `Enum`, which can be set to one of the following: `estwrite`, `estimate`, or `write`. In a container, the default is always the first element. So in this case, the default will be `estwrite`:

```
>>> from nipyre.interfaces import spm
>>> rlgn = spm.Realign()
>>> rlgn.inputs.infile
<undefined>
>>> rlgn.inputs.jobtype
'estwrite'
```

xor and requires Both of these accept a list of trait names. The `xor` metadata reflects mutually exclusive traits, while the `requires` metadata reflects traits that have to be set together. When a `xor`-ed trait is set, all other traits belonging to the list are set to `Undefined`. The function `check_mandatory_inputs` ensures that all requirements (both mandatory and via the `requires` metadata are satisfied). These are also reflected in the `help` function.

copyfile This is metadata for a File or Directory trait that is relevant only in the context of wrapping an interface in a `Node` and `MapNode`. `copyfile` can be set to either `True` or `False`. `False` indicates that contents should be symlinked, while `True` indicates that the contents should be copied over.

min_ver and max_ver These metadata determine if a particular trait will be available when a given version of the underlying interface runs. Note that this check is performed at runtime.:

```
class RealignInputSpec(BaseInterfaceInputSpec):
    jobtype = traits.Enum('estwrite', 'estimate', 'write', min_ver='5',
                          usedefault=True)
```

deprecated and new_name This is metadata for removing or renaming an input field from a spec.:

```
class RealignInputSpec(BaseInterfaceInputSpec):
    jobtype = traits.Enum('estwrite', 'estimate', 'write',
                          deprecated='0.8',
                          desc='one of: estimate, write, estwrite',
                          usedefault=True)
```

In the above example this means that the *jobtype* input is deprecated and will be removed in version 0.8. Deprecation should be set to two versions from current release. Raises *TraitError* after package version crosses the deprecation version.

For inputs that are being renamed, one can specify the new name of the field.:

```
class RealignInputSpec(BaseInterfaceInputSpec):
    jobtype = traits.Enum('estwrite', 'estimate', 'write',
                          deprecated='0.8', new_name='job_type',
                          desc='one of: estimate, write, estwrite',
                          usedefault=True)
    job_type = traits.Enum('estwrite', 'estimate', 'write',
                           desc='one of: estimate, write, estwrite',
                           usedefault=True)
```

In the above example, the *jobtype* field is being renamed to *job_type*. When *new_name* is provided it must exist as a trait, otherwise an exception will be raised.

Note: The version information for *min_ver*, *max_ver* and *deprecated* has to be provided as a string. For example, *min_ver*='0.1'.

CommandLine

argstr The metadata keyword for specifying the format strings for the parameters. This was the *value* string in the *opt_map* dictionaries of Nipytype 0.2 code. If we look at the *FlirtInputSpec*, the *argstr* for the reference file corresponds to the argument string I would need to provide with the command-line version of *flirt*:

```
class FlirtInputSpec(FSLTraitedSpec):
    reference = File(exists = True, argstr = '-ref %s', mandatory = True,
                    position = 1, desc = 'reference file')
```

Required: This metadata is required by all command-line interface classes.

position This metadata is used to specify the position of arguments. Both positive and negative values are accepted. *position* = 0 will position this argument as the first parameter after the command name. *position* = -1 will position this argument as the last parameter, after all other parameters.

genfile If True, the *genfile* metadata specifies that a filename should be generated for this parameter *if-and-only-if* the user did not provide one. The nipytype convention is to automatically generate output filenames when not specified by the user both as a convenience for the user and so the pipeline can easily gather the outputs. Requires *_gen_filename()* method to be implemented. This way should be used if the desired file name is dependent on some runtime variables (such as file name of one of the inputs, or current working directory). In case when it should be fixed it's recommended to just use *usedefault*.

sep For List traits the string with which elements of the list will be joined.

name_source Indicates the list of input fields from which the value of the current File output variable will be drawn. This input field must be the name of a File. Chaining is allowed, meaning that an input field can point to another as *name_source*, which also points as *name_source* to a third field. In this situation, the templates for substitutions are also accumulated.

name_template By default a *%s_generated* template is used to create the output filename. This metadata keyword allows overriding the generated name.

keep_extension Use this and set it True if you want the extension from the input to be kept.

SPM

field name of the structure referred by the SPM job manager

Required: This metadata is required by all SPM-mediated interface classes.

Defining an interface class

Common

When you define an interface class, you will define these attributes and methods:

- `input_spec`: the `InputSpec`
- `output_spec`: the `OutputSpec`
- `_list_outputs()`: Returns a dictionary containing names of generated files that are expected after package completes execution. This is used by `BaseInterface.aggregate_outputs` to gather all output files for the pipeline.

CommandLine

For command-line interfaces:

- `_cmd`: the command-line command

If you used genfile:

- `_gen_filename(name)`: Generate filename, used for filenames that nipy generates as a convenience for users. This is for parameters that are required by the wrapped package, but we're generating from some other parameter. For example, `BET.inputs.outfile` is required by `BET` but we can generate the name from `BET.inputs.infile`. Override this method in subclass to handle.

And optionally:

- `_redirect_x`: If set to `True` it will make Nipy start `Xvfb` before running the interface and redirect `X` output to it. This is useful for commandlines that spawn a graphical user interface.
- `_format_arg(name, spec, value)`: For extra formatting of the input values before passing them to `generic_parse_inputs()` method.

For example this is the class definition for `Flirt`, minus the docstring:

```
class FLIRTInputSpec(FSLCommandInputSpec):
    in_file = File(exists=True, argstr='-in %s', mandatory=True,
                   position=0, desc='input file')
    reference = File(exists=True, argstr='-ref %s', mandatory=True,
                    position=1, desc='reference file')
    out_file = File(argstr='-out %s', desc='registered output file',
                   name_source=['in_file'], name_template='%s_flirt',
                   position=2, hash_files=False)
    out_matrix_file = File(argstr='-omat %s',
                          name_source=['in_file'], keep_extension=True,
                          name_template='%s_flirt.mat',
                          desc='output affine matrix in 4x4 ascii format',
                          position=3, hash_files=False)
    out_log = File(name_source=['in_file'], keep_extension=True,
                  requires=['save_log'],
                  name_template='%s_flirt.log', desc='output log')
    ...

class FLIRTOutputSpec(TraitedSpec):
    out_file = File(exists=True,
                   desc='path/name of registered file (if generated)')
    out_matrix_file = File(exists=True,
                          desc='path/name of calculated affine transform '
                              '(if generated)')
```

```

out_log = File(desc='path/name of output log (if generated)')

class Flirt(FSLCommand):
    _cmd = 'flirt'
    input_spec = FlirtInputSpec
    output_spec = FlirtOutputSpec

```

There are two possible output files `outfile` and `outmatrix`, both of which can be generated if not specified by the user.

Also notice the use of `self._gen_fname()` - a `FSLCommand` helper method for generating filenames (with extensions conforming with `FSLOUTPUTTYPE`).

See also *[How to wrap a command line tool](#)*.

SPM

For SPM-mediated interfaces:

- `_jobtype` and `_jobname`: special names used by the SPM job manager. You can find them by saving your batch job as an `.m` file and looking up the code.

And optionally:

- `_format_arg(name, spec, value)`: For extra formatting of the input values before passing them to `generic_parse_inputs()` method.

Matlab

See *[How to wrap a MATLAB script](#)*.

Python

See *[How to wrap a Python script](#)*.

Undefined inputs

All the inputs and outputs that were not explicitly set (And do not have a `usedefault` flag - see above) will have Undefined value. To check if something is defined you have to explicitly call `isdefined` function (comparing to `None` will not work).

Example of inputs

Below we have an example of using `Bet`. We can see from the help which inputs are mandatory and which are optional, along with the one-line description provided by the `desc` metadata:

```

>>> from nipy.interfaces import fsl
>>> fsl.BET.help()
Inputs
-----

Mandatory:
infile: input file to skull strip

Optional:
args: Additional parameters to the command
center: center of gravity in voxels
environ: Environment variables (default={})
frac: fractional intensity threshold
functional: apply to 4D fMRI data
mask: create binary mask image
mesh: generate a vtk mesh brain surface

```

```
nooutput: Don't generate segmented output
outfile: name of output skull stripped image
outline: create surface outline image
outputtype: None
radius: head radius
reduce_bias: bias field and neck cleanup
skull: create skull image
threshold: apply thresholding to segmented brain image and mask
vertical_gradient: vertical gradient in fractional intensity threshold (-1, 1)
```

Outputs

```
maskfile: path/name of binary brain mask (if generated)
meshfile: path/name of vtk mesh file (if generated)
outfile: path/name of skullstripped file
outlinefile: path/name of outline file (if generated)
```

Here we create a bet object and specify the required input. We then check our inputs to see which are defined and which are not:

```
>>> bet = fsl.BET(infile = 'f3.nii')
>>> bet.inputs
args = <undefined>
center = <undefined>
environ = {'FSLOUTPUTTYPE': 'NIFTI_GZ'}
frac = <undefined>
functional = <undefined>
infile = f3.nii
mask = <undefined>
mesh = <undefined>
nooutput = <undefined>
outfile = <undefined>
outline = <undefined>
outputtype = NIFTI_GZ
radius = <undefined>
reduce_bias = <undefined>
skull = <undefined>
threshold = <undefined>
vertical_gradient = <undefined>
>>> bet.cmdline
'bet f3.nii /Users/cburns/data/nipyne/s1/f3_brain.nii.gz'
```

We also checked the command-line that will be generated when we run the command and can see the generated output filename `f3_brain.nii.gz`.

How to wrap a command line tool

The aim of this section is to describe how external programs and scripts can be wrapped for use in Nipyne either as interactive interfaces or within the workflow/pipeline environment. Currently, there is support for command line executables/scripts and matlab scripts. One can also create pure Python interfaces. The key to defining interfaces is to provide a formal specification of inputs and outputs and determining what outputs are generated given a set of inputs.

Defining inputs and outputs

In Nipyne we use Enthought Traits to define inputs and outputs of the interfaces. This allows to introduce easy type checking. Inputs and outputs are grouped into separate classes (usually suffixed with `InputSpec` and

OutputSpec). For example:

```
class ExampleInputSpec(TraitSpec):
    input_volume = File(desc = "Input volume", exists = True,
                        mandatory = True)
    parameter = traits.Int(desc = "some parameter")

class ExampleOutputSpec(TraitSpec):
    output_volume = File(desc = "Output volume", exists = True)
```

For the Traits (and Nipyte) to work correctly output and input spec has to be inherited from TraitSpec (however, this does not have to be direct inheritance).

Traits (File, Int etc.) have different parameters (called metadata). In the above example we have used the desc metadata which holds human readable description of the input. The mandatory flag forces Nipyte to throw an exception if the input was not set. exists is a special flag that works only for File traits and checks if the provided file exists. More details can be found at [Interface Specifications](#).

The input and output specifications have to be connected to the our example interface class:

```
class Example(Interface):
    input_spec = ExampleInputSpec
    output_spec = ExampleOutputSpec
```

Where the names of the classes grouping inputs and outputs were arbitrary the names of the fields within the interface they are assigned are not (it always has to be input_spec and output_spec). Of course this interface does not do much because we have not specified how to process the inputs and create the outputs. This can be done in many ways.

Command line executable

As with all interfaces command line wrappers need to have inputs defined. Command line input spec has to inherit from CommandLineInputSpec which adds two extra inputs: environ (a dictionary of environmental variables), and args (a string defining extra flags). In addition input spec can define the relation between the inputs and the generated command line. To achieve this we have added two metadata: argstr (string defining how the argument should be formatted) and position (number defining the order of the arguments). For example

```
class ExampleInputSpec(CommandLineSpec):
    input_volume = File(desc = "Input volume", exists = True,
                        mandatory = True, position = 0, argstr="%s")
    parameter = traits.Int(desc = "some parameter", argstr = "--param %d")
```

As you probably noticed the argstr is a printf type string with formatting symbols. For an input defined in InputSpec to be included into the executed commandline argstr has to be included. Additionally inside the main interface class you need to specify the name of the executable by assigning it to the _cmd field. Also the main interface class needs to inherit from `nipyte.interfaces.base.CommandLine`:

```
class Example(CommandLine):
    _cmd = 'my_command'
    input_spec = ExampleInputSpec
    output_spec = ExampleOutputSpec
```

There is one more thing we need to take care of. When the executable finishes processing it will presumably create some output files. We need to know which files to look for, check if they exist and expose them to whatever node would like to use them. This is done by implementing _list_outputs method in the main interface class. Basically what it does is assigning the expected output files to the fields of our output spec:

```
def _list_outputs(self):
    outputs = self.output_spec().get()
```

```

        outputs['output_volume'] = os.path.abspath('name_of_the_file_this_cmd_
↳made.nii')
        return outputs

```

Sometimes the inputs need extra parsing before turning into command line parameters. For example imagine a parameter selecting between three methods: “old”, “standard” and “new”. Imagine also that the command line accept this as a parameter “--method=” accepting 0, 1 or 2. Since we are aiming to make nipyre scripts as informative as possible it’s better to define the inputs as following:

```

class ExampleInputSpec(CommandLineSpec):
    method = traits.Enum("old", "standard", "new", desc = "method",
        argstr="--method=%d")

```

Here we’ve used the Enum trait which restricts input a few fixed options. If we would leave it as it is it would not work since the argstr is expecting numbers. We need to do additional parsing by overloading the following method in the main interface class:

```

def _format_arg(self, name, spec, value):
    if name == 'method':
        return spec.argstr%{"old":0, "standard":1, "new":2}[value]
    return super(Example, self)._format_arg(name, spec, value)

```

Here is a minimalistic interface for the gzip command:

```

from nipyre.interfaces.base import (
    TraitedSpec,
    CommandLineInputSpec,
    CommandLine,
    File
)
import os

class GZipInputSpec(CommandLineInputSpec):
    input_file = File(desc="File", exists=True, mandatory=True, argstr="%s")

class GZipOutputSpec(TraitedSpec):
    output_file = File(desc = "Zip file", exists = True)

class GZipTask(CommandLine):
    input_spec = GZipInputSpec
    output_spec = GZipOutputSpec
    cmd = 'gzip'

    def _list_outputs(self):
        outputs = self.output_spec().get()
        outputs['output_file'] = os.path.abspath(self.inputs.input_file + ".gz
↳")
        return outputs

if __name__ == '__main__':
    zipper = GZipTask(input_file='an_existing_file')
    print zipper.cmdline
    zipper.run()

```

Creating outputs on the fly

In many cases, command line executables will require specifying output file names as arguments on the command line. We have simplified this procedure with three additional metadata terms: `name_source`,

name_template, keep_extension.

For example in the InvWarp class, the inverse_warp parameter is the name of the output file that is created by the routine.

```
class InvWarpInputSpec(FSLCommandInputSpec):
    ...
    inverse_warp = File(argstr='--out=%s', name_source=['warp'],
                        hash_files=False, name_template='%s_inverse',
    ...
```

we add several metadata to inputspec.

name_source indicates which field to draw from, this field must be the name of a File.

hash_files indicates that the input for this field if provided should not be used in computing the input hash for this interface.

name_template (optional) overrides the default _generated suffix

output_name (optional) name of the output (if this is not set same name as the input will be assumed)

keep_extension (optional) if you want the extension from the input or name_template to be kept. The name_template extension always overrides the input extension.

In addition one can add functionality to your class or base class, to allow changing extensions specific to package or interface. This overload function is triggered only if keep_extension is not defined.

```
def self._overload_extension(self, value):
    return value #do whatever you want here with the name
```

Finally, in the outputspec make sure the name matches that of the inputspec.

```
class InvWarpOutputSpec(TraitSpec):
    inverse_warp = File(exists=True,
                        desc=('Name of output file, containing warps that '
                              'are the "reverse" of those in --warp.'))
```

How to wrap a MATLAB script

This is minimal script for wrapping MATLAB code. You should replace the MATLAB code template, and define appropriate inputs and outputs.

Example 1

```
from nipyype.interfaces.matlab import MatlabCommand
from nipyype.interfaces.base import TraitSpec, BaseInterface, \
↳BaseInterfaceInputSpec, File
import os
from string import Template

class ConmapTxt2MatInputSpec(BaseInterfaceInputSpec):
    in_file = File(exists=True, mandatory=True)
    out_file = File('cmatrix.mat', usedefault=True)

class ConmapTxt2MatOutputSpec(TraitSpec):
    out_file = File(exists=True)

class ConmapTxt2Mat(BaseInterface):
    input_spec = ConmapTxt2MatInputSpec
    output_spec = ConmapTxt2MatOutputSpec

    def _run_interface(self, runtime):
```

```
d = dict(in_file=self.inputs.in_file,
        out_file=self.inputs.out_file)
#this is your MATLAB code template
script = Template("""in_file = '$in_file';
out_file = '$out_file';
ConmapTxt2Mat(in_file, out_file);
exit;
""").substitute(d)

# mfile = True will create an .m file with your script and executed.
# Alternatively
# mfile can be set to False which will cause the matlab code to be
# passed
# as a commandline argument to the matlab executable
# (without creating any files).
# This, however, is less reliable and harder to debug
# (code will be reduced to
# a single line and stripped of any comments).

mlab = MatlabCommand(script=script, mfile=True)
result = mlab.run()
return result.runtime

def _list_outputs(self):
    outputs = self._outputs().get()
    outputs['out_file'] = os.path.abspath(self.inputs.out_file)
    return outputs
```

Example 2

By subclassing **MatlabCommand** for your main class, and **MatlabInputSpec** for your input and output spec, you gain access to some useful MATLAB hooks

```
import os
from nipyne.interfaces.base import File, traits
from nipyne.interfaces.matlab import MatlabCommand, MatlabInputSpec

class HelloWorldInputSpec( MatlabInputSpec):
    name = traits.Str( mandatory = True,
                      desc = 'Name of person to say hello to')

class HelloWorldOutputSpec( MatlabInputSpec):
    matlab_output = traits.Str( )

class HelloWorld( MatlabCommand):
    """ Basic Hello World that displays Hello <name> in MATLAB

    Returns
    -----

    matlab_output : capture of matlab output which may be
                    parsed by user to get computation results

    Examples
    -----
```

```

>>> hello = HelloWorld()
>>> hello.inputs.name = 'hello_world'
>>> out = hello.run()
>>> print(out.outputs.matlab_output)
"""
input_spec = HelloWorldInputSpec
output_spec = HelloWorldOutputSpec

def _my_script(self):
    """This is where you implement your script"""
    script = """
    disp('Hello %s Python')
    two = 1 + 1
    """%(self.inputs.name)
    return script

def run(self, **inputs):
    ## inject your script
    self.inputs.script = self._my_script()
    results = super(MatlabCommand, self).run( **inputs)
    stdout = results.runtime.stdout
    # attach stdout to outputs to access matlab results
    results.outputs.matlab_output = stdout
    return results

def _list_outputs(self):
    outputs = self._outputs().get()
    return outputs

```

How to wrap a Python script

This is a minimal pure python interface. As you can see all you need to do is to define inputs, outputs, `_run_interface()` (not `run()`), and `_list_outputs`.

```

from nipy.interfaces.base import BaseInterface, \
    BaseInterfaceInputSpec, traits, File, TraitedSpec
from nipy.utils.filemanip import split_filename

import nibabel as nb
import numpy as np
import os

class SimpleThresholdInputSpec(BaseInterfaceInputSpec):
    volume = File(exists=True, desc='volume to be thresholded', mandatory=True)
    threshold = traits.Float(desc='everything below this value will be set to zero
    ↪',
                           mandatory=True)

class SimpleThresholdOutputSpec(TraitedSpec):
    thresholded_volume = File(exists=True, desc="thresholded volume")

class SimpleThreshold(BaseInterface):

```

```
input_spec = SimpleThresholdInputSpec
output_spec = SimpleThresholdOutputSpec

def _run_interface(self, runtime):
    fname = self.inputs.volume
    img = nb.load(fname)
    data = np.array(img.get_data())

    active_map = data > self.inputs.threshold

    thresholded_map = np.zeros(data.shape)
    thresholded_map[active_map] = data[active_map]

    new_img = nb.Nifti1Image(thresholded_map, img.affine, img.header)
    _, base, _ = split_filename(fname)
    nb.save(new_img, base + '_thresholded.nii')

    return runtime

def _list_outputs(self):
    outputs = self._outputs().get()
    fname = self.inputs.volume
    _, base, _ = split_filename(fname)
    outputs["thresholded_volume"] = os.path.abspath(base + '_thresholded.nii')
    return outputs
```

Working with *nipyre* source code

Contents:

Introduction

These pages describe a [git](#) and [github](#) workflow for the [nipyre](#) project.

There are several different workflows here, for different ways of working with *nipyre*.

This is not a comprehensive [git](#) reference, it's just a workflow for our own project. It's tailored to the [github](#) hosting service. You may well find better or quicker ways of getting stuff done with [git](#), but these should get you started.

For general resources for learning [git](#) see [git resources](#).

Install git

Overview

Debian / Ubuntu	<code>sudo apt-get install git-core</code>
Fedora	<code>sudo yum install git-core</code>
Windows	Download and install msysGit
OS X	Use the git-osx-installer

In detail

See the [git](#) page for the most recent information.

Have a look at the [github](#) install help pages available from [github help](#)

There are good instructions here: http://book.git-scm.com/2_installing_git.html

Following the latest source

These are the instructions if you just want to follow the latest *nipyne* source, but you don't need to do any development for now.

The steps are:

- *Install git*
- get local copy of the git repository from [github](#)
- update local copy from time to time

Get the local copy of the code

From the command line:

```
git clone git://github.com/nipy/nipyne.git
```

You now have a copy of the code tree in the new `nipyne` directory.

Updating the code

From time to time you may want to pull down the latest code. Do this with:

```
cd nipyne
git pull
```

The tree in `nipyne` will now have the latest changes from the initial repository.

Making a patch

You've discovered a bug or something else you want to change in `nipyne` .. — excellent!

You've worked out a way to fix it — even better!

You want to tell us about it — best of all!

The easiest way is to make a *patch* or set of patches. Here we explain how. Making a patch is the simplest and quickest, but if you're going to be doing anything more than simple quick things, please consider following the *Git for development* model instead.

Making patches

Overview

```
# tell git who you are
git config --global user.email you@yourdomain.example.com
git config --global user.name "Your Name Comes Here"
# get the repository if you don't have it
git clone git://github.com/nipy/nipyne.git
# make a branch for your patching
cd nipyne
git branch the-fix-im-thinking-of
git checkout the-fix-im-thinking-of
# hack, hack, hack
# Tell git about any new files you've made
git add somewhere/tests/test_my_bug.py
# commit work in progress as you go
git commit -am 'BF - added tests for Funny bug'
# hack hack, hack
git commit -am 'BF - added fix for Funny bug'
# make the patch files
git format-patch -M -C master
```

Then, send the generated patch files to the [nipyne mailing list](#) — where we will thank you warmly.

In detail

1. Tell `git` who you are so it can label the commits you've made:

```
git config --global user.email you@yourdomain.example.com
git config --global user.name "Your Name Comes Here"
```

2. If you don't already have one, clone a copy of the `nipyre` repository:

```
git clone git://github.com/nipy/nipyre.git
cd nipyre
```

3. Make a 'feature branch'. This will be where you work on your bug fix. It's nice and safe and leaves you with access to an unmodified copy of the code in the main branch:

```
git branch the-fix-im-thinking-of
git checkout the-fix-im-thinking-of
```

4. Do some edits, and commit them as you go:

```
# hack, hack, hack
# Tell git about any new files you've made
git add somewhere/tests/test_my_bug.py
# commit work in progress as you go
git commit -am 'BF - added tests for Funny bug'
# hack hack, hack
git commit -am 'BF - added fix for Funny bug'
```

Note the `-am` options to `commit`. The `m` flag just signals that you're going to type a message on the command line. The `a` flag — you can just take on faith — or see [why the -a flag?](#).

5. When you have finished, check you have committed all your changes:

```
git status
```

6. Finally, make your commits into patches. You want all the commits since you branched from the `master` branch:

```
git format-patch -M -C master
```

You will now have several files named for the commits:

```
0001-BF-added-tests-for-Funny-bug.patch
0002-BF-added-fix-for-Funny-bug.patch
```

Send these files to the [nipyre mailing list](#).

When you are done, to switch back to the main copy of the code, just return to the `master` branch:

```
git checkout master
```

Moving from patching to development

If you find you have done some patches, and you have one or more feature branches, you will probably want to switch to development mode. You can do this with the repository you have.

Fork the `nipyre` repository on [github](#) — *Making your own copy (fork) of nipyre*. Then:

```
# checkout and refresh master branch from main repo
git checkout master
git pull origin master
# rename pointer to main repository to 'upstream'
git remote rename origin upstream
# point your repo to default read / write to your fork on github
```



```
git remote add origin git@github.com:your-user-name/nipype.git
# push up any branches you've made and want to keep
git push origin the-fix-im-thinking-of
```

Then you can, if you want, follow the *Development workflow*.

Git for development

Contents:

Making your own copy (fork) of nipype

You need to do this only once. The instructions here are very similar to the instructions at <http://help.github.com/forking/> — please see that page for more detail. We’re repeating some of it here just to give the specifics for the nipype project, and to suggest some default names.

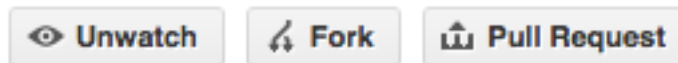
Set up and configure a github account

If you don’t have a [github](#) account, go to the [github](#) page, and make one.

You then need to configure your account to allow write access — see the [Generating SSH keys help on github help](#).

Create your own forked copy of nipype

1. Log into your [github](#) account.
2. Go to the [nipype github](#) home at [nipype github](#).
3. Click on the *fork* button:



Now, after a short pause and some ‘Hardcore forking action’, you should find yourself at the home page for your own forked copy of [nipype](#).

Set up your fork

First you follow the instructions for *Making your own copy (fork) of nipype*.

Overview

```
git clone git@github.com:your-user-name/nipype.git
cd nipype
git remote add upstream git://github.com/nipy/nipype.git
```

In detail

Clone your fork

1. Clone your fork to the local computer with `git clone git@github.com:your-user-name/nipype.git`

- Investigate. Change directory to your new repo: `cd nipyne`. Then `git branch -a` to show you all branches. You'll get something like:

```
* master
remotes/origin/master
```

This tells you that you are currently on the `master` branch, and that you also have a remote connection to `origin/master`. What remote repository is `remote/origin`? Try `git remote -v` to see the URLs for the remote. They will point to your [github](#) fork.

Now you want to connect to the upstream [nipyne github](#) repository, so you can merge in changes from trunk.

Linking your repository to the upstream repo

```
cd nipyne
git remote add upstream git://github.com/nipy/nipyne.git
```

`upstream` here is just the arbitrary name we're using to refer to the main [nipyne](#) repository at [nipyne github](#). Note that we've used `git://` for the URL rather than `git@`. The `git://` URL is read only. This means we that we can't accidentally (or deliberately) write to the upstream repo, and we are only going to use it to merge into our own code.

Just for your own satisfaction, show yourself that you now have a new 'remote', with `git remote -v` show, giving you something like:

```
upstream      git://github.com/nipy/nipyne.git (fetch)
upstream      git://github.com/nipy/nipyne.git (push)
origin        git@github.com:your-user-name/nipyne.git (fetch)
origin        git@github.com:your-user-name/nipyne.git (push)
```

Configure git

Overview

Your personal `git` configurations are saved in the `.gitconfig` file in your home directory. Here is an example `.gitconfig` file:

```
[user]
  name = Your Name
  email = you@yourdomain.example.com

[alias]
  ci = commit -a
  co = checkout
  st = status -a
  stat = status -a
  br = branch
  wdifff = diff --color-words

[core]
  editor = vim

[merge]
  summary = true
```

You can edit this file directly or you can use the `git config --global` command:

```
git config --global user.name "Your Name"
git config --global user.email you@yourdomain.example.com
git config --global alias.ci "commit -a"
git config --global alias.co checkout
git config --global alias.st "status -a"
git config --global alias.stat "status -a"
git config --global alias.br branch
git config --global alias.wdiff "diff --color-words"
git config --global core.editor vim
git config --global merge.summary true
```

To set up on another computer, you can copy your `~/ .gitconfig` file, or run the commands above.

In detail

user.name and user.email

It is good practice to tell `git` who you are, for labeling any changes you make to the code. The simplest way to do this is from the command line:

```
git config --global user.name "Your Name"
git config --global user.email you@yourdomain.example.com
```

This will write the settings into your git configuration file, which should now contain a user section with your name and email:

```
[user]
  name = Your Name
  email = you@yourdomain.example.com
```

Of course you'll need to replace `Your Name` and `you@yourdomain.example.com` with your actual name and email address.

Aliases

You might well benefit from some aliases to common commands.

For example, you might well want to be able to shorten `git checkout` to `git co`. Or you may want to alias `git diff --color-words` (which gives a nicely formatted output of the diff) to `git wdiff`

The following `git config --global` commands:

```
git config --global alias.ci "commit -a"
git config --global alias.co checkout
git config --global alias.st "status -a"
git config --global alias.stat "status -a"
git config --global alias.br branch
git config --global alias.wdiff "diff --color-words"
```

will create an alias section in your `.gitconfig` file with contents like this:

```
[alias]
  ci = commit -a
  co = checkout
  st = status -a
  stat = status -a
  br = branch
  wdiff = diff --color-words
```

Editor

You may also want to make sure that your editor of choice is used

```
git config --global core.editor vim
```

Merging

To enforce summaries when doing merges (~/.gitconfig file again):

```
[merge]
  log = true
```

Or from the command line:

```
git config --global merge.log true
```

Development workflow

You already have your own forked copy of the [nipyre](#) repository, by following *Making your own copy (fork) of nipyre*, *Set up your fork*, and you have configured [git](#) by following *Configure git*.

Workflow summary

- Keep your `master` branch clean of edits that have not been merged to the main [nipyre](#) development repo. Your `master` then will follow the main [nipyre](#) repository.
- Start a new *feature branch* for each set of edits that you do.
- If you can avoid it, try not to merge other branches into your feature branch while you are working.
- Ask for review!

This way of working really helps to keep work well organized, and in keeping history as clear as possible. See — for example — [linux git workflow](#).

Making a new feature branch

```
git branch my-new-feature
git checkout my-new-feature
```

Generally, you will want to keep this also on your public [github](#) fork of [nipyre](#). To do this, you `git push` this new branch up to your [github](#) repo. Generally (if you followed the instructions in these pages, and by default), `git` will have a link to your [github](#) repo, called `origin`. You push up to your own repo on [github](#) with:

```
git push origin my-new-feature
```

In `git >1.7` you can ensure that the link is correctly set by using the `--set-upstream` option:

```
git push --set-upstream origin my-new-feature
```

From now on `git` will know that `my-new-feature` is related to the `my-new-feature` branch in the [github](#) repo.

The editing workflow

Overview

```
# hack hack
git add my_new_file
git commit -am 'NF - some message'
git push
```

In more detail

1. Make some changes
2. See which files have changed with `git status` (see [git status](#)). You'll see a listing like this one:

```
# On branch ny-new-feature
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#   modified:   README
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#   INSTALL
no changes added to commit (use "git add" and/or "git commit -a")
```

3. Check what the actual changes are with `git diff` ([git diff](#)).
4. Add any new files to version control `git add new_file_name` (see [git add](#)).
5. To commit all modified files into the local copy of your repo., do `git commit -am 'A commit message'`. Note the `-am` options to commit. The `m` flag just signals that you're going to type a message on the command line. The `a` flag — you can just take on faith — or see [why the -a flag?](#) — and the helpful use-case description in the [tangled working copy problem](#). The [git commit](#) manual page might also be useful.
6. To push the changes up to your forked repo on [github](#), do a `git push` (see [git push](#)).

Asking for code review

1. Go to your repo URL — e.g. <http://github.com/your-user-name/nipyre>.
2. Click on the *Branch list* button:



3. Click on the *Compare* button for your feature branch — here `my-new-feature`:

NAME	STATE
my-new-feature Last updated 18 minutes ago	0 ahead 0 behind

Compare

4. If asked, select the *base* and *comparison* branch names you want to compare. Usually these will be `master` and `my-new-feature` (where that is your feature branch name).
5. At this point you should get a nice summary of the changes. Copy the URL for this, and post it to the [nipyre mailing list](#), asking for review. The URL will look something like: <http://github.com/your-user-name/nipyre/compare/master...my-new-feature>. There's an example at <http://github.com/matthew-brett/nipy/compare/master...find-install-data> See: <http://github.com/blog/612-introducing-github-compare-view> for more detail.

The generated comparison, is between your feature branch `my-new-feature`, and the place in `master` from which you branched `my-new-feature`. In other words, you can keep updating `master` without interfering with the output from the comparison. More detail? Note the three dots in the URL above (`master...my-new-feature`).

Two vs three dots

Imagine a series of commits A, B, C, D... Imagine that there are two branches, *topic* and *master*. You branched *topic* off *master* when *master* was at commit 'E'. The graph of the commits looks like this:

```
      A---B---C topic
      /
D---E---F---G master
```

Then:

```
git diff master..topic
```

will output the difference from G to C (i.e. with effects of F and G), while:

```
git diff master...topic
```

would output just differences in the topic branch (i.e. only A, B, and C).¹

Asking for your changes to be merged with the main repo

When you are ready to ask for the merge of your code:

1. Go to the URL of your forked repo, say `http://github.com/your-user-name/nipyre.git`.
2. Click on the 'Pull request' button:



Enter a message; we suggest you select only `nipyre` as the recipient. The message will go to the [nipyre mailing list](#). Please feel free to add others from the list as you like.

Merging from trunk

This updates your code from the upstream `nipyre` github repo.

Overview

```
# go to your master branch
git checkout master
# pull changes from github
git fetch upstream
# merge from upstream
git merge upstream/master
```

¹ Thanks to Yarik Halchenko for this explanation.

In detail

We suggest that you do this only for your `master` branch, and leave your ‘feature’ branches unmerged, to keep their history as clean as possible. This makes code review easier:

```
git checkout master
```

Make sure you have done *Linking your repository to the upstream repo*.

Merge the upstream code into your current development by first pulling the upstream repo to a copy on your local machine:

```
git fetch upstream
```

then merging into your current branch:

```
git merge upstream/master
```

Deleting a branch on github

```
git checkout master
# delete branch locally
git branch -D my-unwanted-branch
# delete branch on github
git push origin :my-unwanted-branch
```

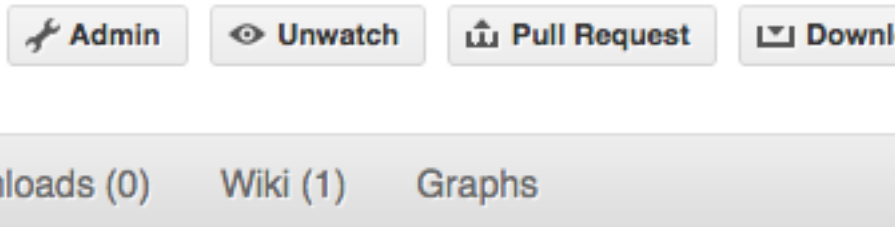
(Note the colon `:` before `test-branch`. See also: <http://github.com/guides/remove-a-remote-branch>)

Several people sharing a single repository

If you want to work on some stuff with other people, where you are all committing into the same repository, or even the same branch, then just share it via [github](#).

First fork nipyre into your account, as from *Making your own copy (fork) of nipyre*.

Then, go to your forked repository github page, say <http://github.com/your-user-name/nipyre>. Click on the ‘Admin’ button, and add anyone else to the repo as a collaborator:



Now all those people can do:

```
git clone git@github.com:your-user-name/nipyre.git
```

Remember that links starting with `git@` use the ssh protocol and are read-write; links starting with `git://` are read-only.

Your collaborators can then commit directly into that repo with the usual:

```
git commit -am 'ENH - much better code'
git push origin master # pushes directly into your repo
```

Exploring your repository

To see a graphical representation of the repository branches and commits:

```
gitk --all
```

To see a linear list of commits for this branch:

```
git log
```

You can also look at the [network graph visualizer](#) for your [github](#) repo.

git resources

Tutorials and summaries

- [github help](#) has an excellent series of how-to guides.
- [learn.github](#) has an excellent series of tutorials
- The [pro git book](#) is a good in-depth book on git.
- A [git cheat sheet](#) is a page giving summaries of common commands.
- The [git user manual](#)
- The [git tutorial](#)
- The [git community book](#)
- [git ready](#) — a nice series of tutorials
- [git casts](#) — video snippets giving git how-tos.
- [git magic](#) — extended introduction with intermediate detail
- The [git parable](#) is an easy read explaining the concepts behind git.
- Our own [git foundation](#) expands on the [git parable](#).
- Fernando Perez' [git page](#) — [Fernando's git page](#) — many links and tips
- A good but technical page on [git concepts](#)
- [git svn crash course](#): [git](#) for those of us used to [subversion](#)

Advanced git workflow

There are many ways of working with [git](#); here are some posts on the rules of thumb that other projects have come up with:

- Linus Torvalds on [git management](#)
- Linus Torvalds on [linux git workflow](#) . Summary; use the git tools to make the history of your edits as clean as possible; merge from upstream edits as little as possible in branches where you are doing active development.

Manual pages online

You can get these on your own machine with (e.g) `git help push` or (same thing) `git push --help`, but, for convenience, here are the online manual pages for some common commands:

- [git add](#)
- [git branch](#)
- [git checkout](#)
- [git clone](#)
- [git commit](#)
- [git config](#)
- [git diff](#)
- [git log](#)
- [git pull](#)
- [git push](#)
- [git remote](#)
- [git status](#)

Architecture (discussions from 2009)

This section reflects notes and discussion between developers during the start of the nipy project in 2009.

Design Guidelines

These are guidelines that the core nipy developers have agreed on:

Interfaces should keep all parameters affecting construction of the appropriate command in the “input” bunch. The `.run()` method of an Interface should include all required inputs as explicitly named parameters, and they should take a default value of `None`.

Any Interface should at a minimum support `cwd` as a command-line argument to `.run()`. This may be accomplished by allowing `cwd` as an element of the input Bunch, or handled as a separate case.

Relatedly, any Interface should output all files to `cwd` if it is set, and otherwise to `os.getcwd()` (or equivalent).

We need to decide on a consistent policy towards the maintenance of paths to files. It seems like the best strategy might be to do absolute (`os.realpath?`) filenames by default, allowing for relative paths by explicitly including something that doesn’t start with a `’/’`. This could include `’.’` in some sort of path-spec.

Class attributes should never be modified by an instance of that class. And probably not ever.

Providing for Provenance

The following is a specific discussion that should be thought out and more generally applied to the way we handle auto-generation / or “sourcing” of settings in an interface.

There are two possible sources (at a minimum) from which the interface instance could obtain “outputtype” - itself, or `FSLInfo`. Currently, the outputtype gets read from `FSLInfo` if `self.outputtype` (er, `_outputtype?`) is `None`.

In the case of other `opt_map` specifications, there are defaults that get specified if the value is `None`. For example output filenames are often auto-generated. If you look at the code for `fsl.Bet` for example, there is no way for the outfile to get picked up at the pipeline level, because it is a transient variable. This is OK, as the generation of the outfile name is contingent ONLY on inputs which ARE available to the pipeline machinery (i.e., via inspection of the `Bet` instance’s attributes).

However, with outputtype, we are in a situation in which “autogeneration” incorporates potentially transient information external to the instance itself. Thus, some care needs to be taken in always ensuring this information is hashable.

Design Principles

These are (currently) Dav Clark’s best guess at what the group might agree on:

It should be very easy to figure out what was done by the pipeline.

Code should support relocatability - this could be via URIs, relative paths or potentially other mechanisms.

Unless otherwise called for, code should be thread safe, just in case.

The pipeline should make it easy to change aspects of an analysis with minimal recomputation, downloading, etc. (This is not the case currently - any change will overwrite the old node). Also, the fact that multiple files get rolled into a single node is problematic for similar reasons. E.g. - `node([file1 ... file100])` will get recomputed if we add only one file!

However, it should also be easy to identify and delete things you don’t need anymore.

Pipelines and bits of pipelines should be easy to share.

Things that are the same should be called the same thing in most places. For interfaces that have an obvious meaning for the terms, “infiles” and “outfile(s)”. If a file is in both the inputs and outputs, it should be called the same thing in both places. If it is produced by one interface and consumed by another, same thing should be used.

Discussions

Auto-generated filenames

In refactoring the inputs in the traitlets branch I'm working through the different ways that filenames are generated and want to make sure the interface is consistent. The notes below are all using `fsl.Bet` as that's the first class we're Traiting. Other interface classes may handle this differently, but should agree on a convention and apply it across all Interfaces (if possible).

Current Rules

These rules are for `fsl.Bet`, but it appears they are the same for all `fsl` and `spm` Interfaces.

`Bet` has two mandatory parameters, `infile` and `outfile`. These are the rules for how they are handled in different use cases.

1. If `infile` or `outfile` are absolute paths, they are used as-is and never changed. This allows users to override any filename/path generation.
2. If `outfile` is not specified, a filename is generated.
3. Generated filenames (at least for `outfile`) are based on:
 - `infile`, the filename minus the extensions.
 - A suffix specified by the Interface. For example `Bet` uses `_brain` suffix.
 - The current working directory, `os.getcwd()`. Example:
If `infile == 'foo.nii'` and the `cwd` is `/home/cburns` then generated `outfile` for `Bet` will be `/home/cburns/foo_brain.nii.gz`
4. If `outfile` is not an absolute path, for instance just a filename, the absolute path is generated using `os.path.realpath`. This absolute path is needed to make sure the packages (`Bet` in this case) write the output file to a location of our choosing. The generated absolute path is only used in the `cmdline` at runtime and does not overwrite the class attr `self.inputs.outfile`. It is generated only when the `cmdline` is invoked.

Walking through some examples

In this example we assign `infile` directly but `outfile` is generated in `Bet._parse_inputs` based on `infile`. The generated `outfile` is only used in the `cmdline` at runtime and not stored in `self.inputs.outfile`. This seems correct.

```
In [15]: from nipy.interfaces import fsl

In [16]: mybet = fsl.Bet()

In [17]: mybet.inputs.infile = 'foo.nii'

In [18]: res = mybet.run()

In [19]: res.runtime.cmdline
Out[19]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipy/trunk/nipy/interfaces/
↳tests/foo_brain.nii.gz'

In [21]: mybet.inputs
Out[21]: Bunch(center=None, flags=None, frac=None, functional=None,
infile='foo.nii', mask=None, mesh=None, nooutput=None, outfile=None,
outline=None, radius=None, reduce_bias=None, skull=None, threshold=None,
verbose=None, vertical_gradient=None)

In [24]: mybet.cmdline
Out[24]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipy/trunk/nipy/interfaces/
↳tests/foo_brain.nii.gz'

In [25]: mybet.inputs.outfile
```

```
In [26]: mybet.inputs.infile
Out[26]: 'foo.nii'
```

We get the same behavior here when we assign infile at initialization:

```
In [28]: mybet = fsl.Bet(infile='foo.nii')

In [29]: mybet.cmdline
Out[29]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
↳tests/foo_brain.nii.gz'

In [30]: mybet.inputs
Out[30]: Bunch(center=None, flags=None, frac=None, functional=None,
infile='foo.nii', mask=None, mesh=None, nooutput=None, outfile=None,
outline=None, radius=None, reduce_bias=None, skull=None, threshold=None,
verbose=None, vertical_gradient=None)

In [31]: res = mybet.run()

In [32]: res.runtime.cmdline
Out[32]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
↳tests/foo_brain.nii.gz'
```

Here we specify absolute paths for both infile and outfile. The command line's look as expected:

```
In [53]: import os

In [54]: mybet = fsl.Bet()

In [55]: mybet.inputs.infile = os.path.join('/Users/cburns/tmp/junk', 'foo.nii')
In [56]: mybet.inputs.outfile = os.path.join('/Users/cburns/tmp/junk', 'bar.nii')

In [57]: mybet.cmdline
Out[57]: 'bet /Users/cburns/tmp/junk/foo.nii /Users/cburns/tmp/junk/bar.nii'

In [58]: res = mybet.run()

In [59]: res.runtime.cmdline
Out[59]: 'bet /Users/cburns/tmp/junk/foo.nii /Users/cburns/tmp/junk/bar.nii'
```

Here passing in a new outfile in the run method will update mybet.inputs.outfile to the passed in value. Should this be the case?

```
In [110]: mybet = fsl.Bet(infile='foo.nii', outfile='bar.nii')

In [111]: mybet.inputs.outfile
Out[111]: 'bar.nii'

In [112]: mybet.cmdline
Out[112]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
↳tests/bar.nii'

In [113]: res = mybet.run(outfile = os.path.join('/Users/cburns/tmp/junk', 'not_
↳bar.nii'))

In [114]: mybet.inputs.outfile
Out[114]: '/Users/cburns/tmp/junk/not_bar.nii'
```

```
In [115]: mybet.cmdline
Out[115]: 'bet foo.nii /Users/cburns/tmp/junk/not_bar.nii'
```

In this case we provide `outfile` but not as an absolute path, so the absolute path is generated and used for the `cmdline` when run, but `mybet.inputs.outfile` is not updated with the absolute path.

```
In [74]: mybet = fsl.Bet(infile='foo.nii', outfile='bar.nii')

In [75]: mybet.inputs.outfile
Out[75]: 'bar.nii'

In [76]: mybet.cmdline
Out[76]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
↳tests/bar.nii'

In [77]: res = mybet.run()

In [78]: res.runtime.cmdline
Out[78]: 'bet foo.nii /Users/cburns/src/nipy-sf/nipype/trunk/nipype/interfaces/
↳tests/bar.nii'

In [80]: res.interface.inputs.outfile
Out[80]: 'bar.nii'
```

W3C PROV support

Overview

We're using the the [W3C PROV data model](#) to capture and represent provenance in Nipype.

For an overview see:

[PROV-DM overview](#)

Each interface writes out a `provenance.json` (currently `prov.json`) or `provenance.rdf` (if `rdflib` is available) file. The workflow engine can also write out a provenance of the workflow if instructed.

This is very much an experimental feature as we continue to refine how exactly the provenance should be stored and how such information can be used for reporting or reconstituting workflows. By default provenance writing is disabled for the 0.9 release, to enable insert the following code at the top of your script:

```
>>> from nipype import config
>>> config.enable_provenance()
```

Software using Nipype

Configurable Pipeline for the Analysis of Connectomes (C-PAC)

C-PAC is an open-source software pipeline for automated preprocessing and analysis of resting-state fMRI data. C-PAC builds upon a robust set of existing software packages including AFNI, FSL, and ANTS, and makes it easy for both novice users and experts to explore their data using a wide array of analytic tools. Users define analysis pipelines by specifying a combination of preprocessing options and analyses to be run on an arbitrary number of subjects. Results can then be compared across groups using the integrated group statistics feature. C-PAC makes extensive use of Nipype Workflows and Interfaces.

BRAINSTools

[BRAINSTools](#) is a suite of tools for medical image processing focused on brain analysis.

Brain Imaging Pipelines (BIPs)

BIPs is a set of predefined Nipype workflows coupled with a graphical interface and ability to save and share workflow configurations. It provides both Nipype Workflows and Interfaces.

BROCCOLI

BROCCOLI is a piece of software for fast fMRI analysis on many core CPUs and GPUs. It provides Nipype Interfaces.

Forward

Forward is set of tools simplifying the preparation of accurate electromagnetic head models for EEG forward modeling. It uses Nipype Workflows and Interfaces.

Limbo

Limbo is a toolbox for finding brain regions that are neither significantly active nor inactive, but rather “in limbo”. It was build using custom Nipype Interfaces and Workflows.

Lyman

Lyman is a high-level ecosystem for analyzing task based fMRI neuroimaging data using open-source software. It aims to support an analysis workflow that is powerful, flexible, and reproducible, while automating as much of the processing as possible. It is build upon Nipype Workflows and Interfaces.

Medimsight

Medimsight is a commercial service medical imaging cloud platform. It uses Nipype to interface with various neuroimaging software.

MIA

MIA MIA is a a toolkit for gray scale medical image analysis. It provides Nipype interfaces for easy integration with other software.

Mindboggle

Mindboggle software package automates shape analysis of anatomical labels and features extracted from human brain MR image data. Mindboggle can be run as a single command, and can be easily installed as a cross-platform virtual machine for convenience and reproducibility of results. Behind the scenes, open source Python and C++ code run within a Nipype pipeline framework.

OpenfMRI

OpenfMRI is a repository for task based fMRI datasets. It uses Nipype for automated analysis of the deposited data.

serial functional Diffusion Mapping (sfDM)

‘sfDM <<http://github.com/PIRCImagingTools/sfDM>>’ is a software package for looking at changes in diffusion profiles of different tissue types across time. It uses Nipype to process the data.

The Stanford CNI MRS Library (SMAL)

SMAL is a library providing algorithms and methods to read and analyze data from Magnetic Resonance Spectroscopy (MRS) experiments. It provides an API for fitting models of the spectral line-widths of several different molecular species, and quantify their relative abundance in human brain tissue. SMAL uses Nipyre Workflows and Interfaces.

tract_querier

tract_querier is a White Matter Query Language tool. It provides Nipyre interfaces.

Testing nipyre

In order to ensure the stability of each release of Nipyre, the project uses two continuous integration services: **CircleCI** and **Travis CI**. If both batteries of tests are passing, the following badges should be shown in green color:

Installation for developers

To check out the latest development version:

```
git clone https://github.com/nipy/nipyre.git
```

After cloning:

```
cd nipyre
pip install -r requirements.txt
python setup.py develop
```

or:

```
cd nipyre
pip install -r requirements.txt
pip install -e .[tests]
```

Test implementation

Nipyre testing framework is built upon **pytest**. By the time these guidelines are written, Nipyre implements 17638 tests.

After installation in developer mode, the tests can be run with the following simple command at the root folder of the project

```
make tests
```

If make is not installed in the system, it is possible to run the tests using:

```
py.test --doctest-modules --cov=nipyre nipyre
```

A successful test run should complete in 10-30 minutes and end with something like:

```
-----
2445 passed, 41 skipped, 7 xfailed in 1277.66 seconds
```

No test should fail (unless you're missing a dependency). If the `SUBJECTS_DIR`` environment variable is not set, some FreeSurfer related tests will fail. If any of the tests failed, please report them on our [bug tracker](#).

On Debian systems, set the following environment variable before running tests:

```
export MATLABCMD=$pathtomatlabdir/bin/$platform/MATLAB
```

where `$pathtomatlabdir` is the path to your matlab installation and `$platform` is the directory referring to x86 or x64 installations (typically `glnxa64` on 64-bit installations).

Skip tests

Nipyte will skip some tests depending on the currently available software and data dependencies. Installing software dependencies and downloading the necessary data will reduce the number of skip tests.

Some tests in Nipyte make use of some images distributed within the [FSL course data](#). This reduced version of the package can be downloaded [here](#). To enable the tests depending on these data, just unpack the targz file and set the `FSL_COURSE_DATA` environment variable to point to that folder.

Xfail tests

Some tests are expect to fail until the code will be changed or for other reasons.

Avoiding any MATLAB calls from testing

On unix systems, set an empty environment variable:

```
export NIPYPE_NO_MATLAB=
```

This will skip any tests that require matlab.

Testing Nipyte using Docker

As of `nipyte-0.13`, Nipyte is tested inside Docker containers. First, install the [Docker Engine](#). Nipyte has one base docker image called `nipyte/base:latest`, and several additional test images for various Python versions. The base nipyte image is built as follows:

```
cd path/to/nipyte/
docker build -t nipyte/base:latest -f docker/base.Dockerfile .
```

This base image contains several useful tools (FreeSurfer, AFNI, FSL, ANTs, etc.), but not nipyte. It is possible to fetch a built image from the latest master branch of nipyte using:

```
docker run -it --rm nipyte/nipyte:master
```

The docker run command will then open the container and offer a bash shell for the developer. For building a container for running nipyte in Python 3.6:

```
cd path/to/nipyte/
docker build -f Dockerfile -t nipyte/nipyte_test:py36 .
docker run -it --rm -e FSL_COURSE_DATA="/root/examples/nipyte-fsl_course_data" \
    -v ~/examples:/root/examples:ro \
    -v ~/scratch:/scratch \
    -w /root/src/nipyte \
    nipyte/nipyte_test:py36 /usr/bin/run_pytests.sh
```

The last examples assume that the example data is downloaded into `~/examples` and the `~/scratch` folder will be created if it does not exist previously.

Interfaces, Workflows and Examples

- Workflows

`get_flirt_schedule()`

[Link to code](#)

workflows.dmri.camino.connectivity_mapping

create_connectivity_pipeline()

[Link to code](#)

Creates a pipeline that does the same connectivity processing as in the *dmri: Connectivity - Camino, CMTK, FreeSurfer* example script. Given a subject id (and completed Freesurfer reconstruction) diffusion-weighted image, b-values, and b-vectors, the workflow will return the subject's connectome as a Connectome File Format (CFF) file for use in Connectome Viewer (<http://www.cmtk.org>).

Example

```
>>> from nipy.workflow.dmri.camino.connectivity_mapping import create_
↳ connectivity_pipeline
>>> conmapper = create_connectivity_pipeline("nipy_conmap")
>>> conmapper.inputs.inputnode.subjects_dir = '.'
>>> conmapper.inputs.inputnode.subject_id = 'subj1'
>>> conmapper.inputs.inputnode.dwi = 'data.nii.gz'
>>> conmapper.inputs.inputnode.bvecs = 'bvecs'
>>> conmapper.inputs.inputnode.bvals = 'bvals'
>>> conmapper.run()
```

Inputs:

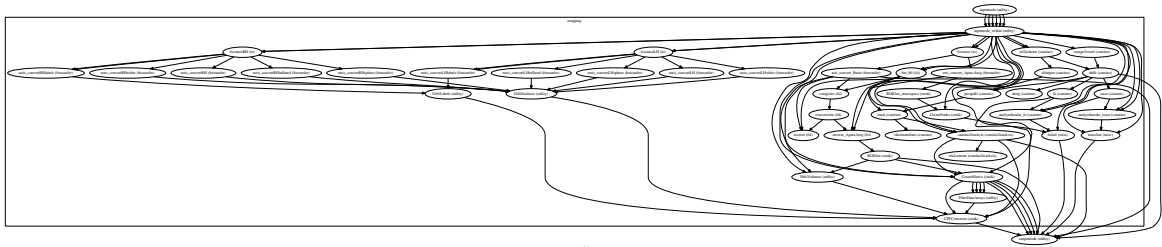
```
inputnode.subject_id
inputnode.subjects_dir
inputnode.dwi
inputnode.bvecs
inputnode.bvals
inputnode.resolution_network_file
```

Outputs:

```
outputnode.connectome
outputnode.cmatrix
outputnode.gpickled_network
outputnode.fa
```

```
outputnode.struct
outputnode.trace
outputnode.tracts
outputnode.tensors
```

Graph



workflows.dmri.camino.diffusion

create_camino_dti_pipeline()

[Link to code](#)

Creates a pipeline that does the same diffusion processing as in the `../users/examples/dmri_camino_dti` example script. Given a diffusion-weighted image, b-values, and b-vectors, the workflow will return the tractography computed from diffusion tensors and from PICO probabilistic tractography.

Example

```
>>> import os
>>> nipyype_camino_dti = create_camino_dti_pipeline("nipyype_camino_dti")
>>> nipyype_camino_dti.inputs.inputnode.dwi = os.path.abspath('dwi.nii')
>>> nipyype_camino_dti.inputs.inputnode.bvecs = os.path.abspath('bvecs')
>>> nipyype_camino_dti.inputs.inputnode.bvals = os.path.abspath('bvals')
>>> nipyype_camino_dti.run()
```

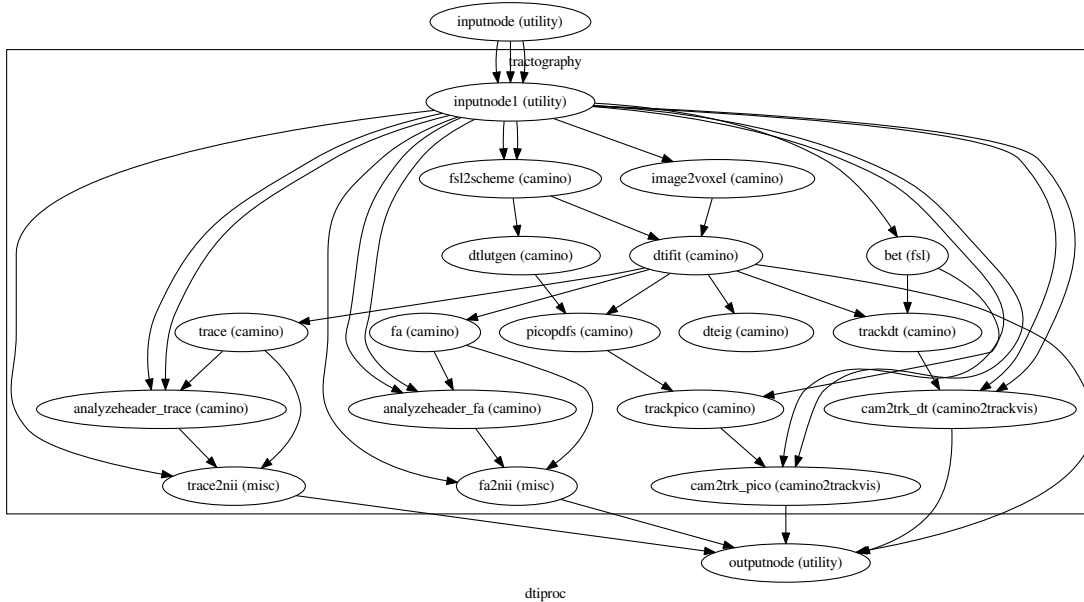
Inputs:

```
inputnode.dwi
inputnode.bvecs
inputnode.bvals
```

Outputs:

```
outputnode.fa
outputnode.trace
outputnode.tracts_pico
outputnode.tracts_dt
outputnode.tensors
```

Graph



workflows.dmri.camino.group_connectivity

create_group_connectivity_pipeline()

[Link to code](#)

Creates a pipeline that performs basic Camino structural connectivity processing on groups of subjects. Given a diffusion-weighted image, and text files containing the associated b-values and b-vectors, the workflow will return each subjects' connectomes in a Connectome File Format (CFF) file, for use in Connectome Viewer (<http://www.cmtk.org>).

Example

```
>>> import nipype.interfaces.freesurfer as fs
>>> import nipype.workflows.dmri.camino.group_connectivity as groupwork
>>> subjects_dir = '.'
>>> data_dir = '.'
>>> output_dir = '.'
>>> fs.FSCommand.set_default_subjects_dir(subjects_dir)
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> template_args = dict(dwi=[['subject_id', 'dwi']], bvecs=[['subject_id', 'bvecs
↪']], bvals=[['subject_id', 'bvals']])
>>> group_id = 'group1'
>>> l1pipeline = groupwork.create_group_connectivity_pipeline(group_list, group_
↪id, data_dir, subjects_dir, output_dir, template_args)
>>> l1pipeline.run()
```

Inputs:

```
group_list: Dictionary of subject lists, keyed by group name
group_id: String containing the group name
data_dir: Path to the data directory
subjects_dir: Path to the Freesurfer 'subjects' directory
output_dir: Path for the output files
template_args_dict: Dictionary of template arguments for the connectivity_
↳ pipeline datasources
    e.g.      info = dict(dwi=[['subject_id', 'dwi']],
                        bvecs=[['subject_id', 'bvecs']],
                        bvals=[['subject_id', 'bvals']])
```

workflows.dmri.connectivity.group_connectivity

concatcsv()

[Link to code](#)

This function will concatenate two “comma-separated value” text files, but remove the first row (usually column headers) from all but the first file.

create_average_networks_by_group_workflow()

[Link to code](#)

Creates a fourth-level pipeline to average the networks for two groups and merge them into a single CFF file. This pipeline will also output the average networks in .gexf format, for visualization in other graph viewers, such as Gephi.

Example

```
>>> import nipyne.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipyne.testing import example_data
>>> subjects_dir = '.'
>>> data_dir = '.'
>>> output_dir = '.'
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> l4pipeline = groupwork.create_average_networks_by_group_workflow(group_list,
↳ data_dir, subjects_dir, output_dir)
>>> l4pipeline.run()
```

Inputs:

```
group_list: Dictionary of subject lists, keyed by group name
data_dir: Path to the data directory
subjects_dir: Path to the Freesurfer 'subjects' directory
output_dir: Path for the output files
title: String to use as a title for the output merged CFF file (default 'group')
```

create_merge_group_network_results_workflow()

[Link to code](#)

Creates a third-level pipeline to merge the Connectome File Format (CFF) outputs from each group and combines them into a single CFF file for each group. This version of the third-level pipeline also concatenates the comma-separated value files for the NetworkX metrics and the connectivity matrices into single files.

Example

```
>>> import nipype.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipype.testing import example_data
>>> subjects_dir = '.'
>>> data_dir = '.'
>>> output_dir = '.'
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> l3pipeline = groupwork.create_merge_group_network_results_workflow(group_list,
↳ data_dir, subjects_dir, output_dir)
>>> l3pipeline.run()
```

Inputs:

```
group_list: Dictionary of subject lists, keyed by group name
data_dir: Path to the data directory
subjects_dir: Path to the Freesurfer 'subjects' directory
output_dir: Path for the output files
title: String to use as a title for the output merged CFF file (default 'group')
```

create_merge_group_networks_workflow()

[Link to code](#)

Creates a third-level pipeline to merge the Connectome File Format (CFF) outputs from each group and combines them into a single CFF file for each group.

Example

```
>>> import nipype.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipype.testing import example_data
>>> subjects_dir = '.'
>>> data_dir = '.'
>>> output_dir = '.'
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> l3pipeline = groupwork.create_merge_group_networks_workflow(group_list, data_
↳ dir, subjects_dir, output_dir)
>>> l3pipeline.run()
```

Inputs:

```
group_list: Dictionary of subject lists, keyed by group name
data_dir: Path to the data directory
subjects_dir: Path to the Freesurfer 'subjects' directory
output_dir: Path for the output files
title: String to use as a title for the output merged CFF file (default 'group')
```

create_merge_network_results_by_group_workflow()

[Link to code](#)

Creates a second-level pipeline to merge the Connectome File Format (CFF) outputs from the group-level MR-trix structural connectivity processing pipeline into a single CFF file for each group.

Example

```
>>> import nipyne.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipyne.testing import example_data
>>> subjects_dir = '.'
>>> data_dir = '.'
>>> output_dir = '.'
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> group_id = 'group1'
>>> l2pipeline = groupwork.create_merge_network_results_by_group_workflow(group_
↳ list, group_id, data_dir, subjects_dir, output_dir)
>>> l2pipeline.run()
```

Inputs:

```
group_list: Dictionary of subject lists, keyed by group name
group_id: String containing the group name
data_dir: Path to the data directory
subjects_dir: Path to the Freesurfer 'subjects' directory
output_dir: Path for the output files
```

create_merge_networks_by_group_workflow()

[Link to code](#)

Creates a second-level pipeline to merge the Connectome File Format (CFF) outputs from the group-level MR-trix structural connectivity processing pipeline into a single CFF file for each group.

Example

```
>>> import nipyne.workflows.dmri.connectivity.group_connectivity as groupwork
>>> from nipyne.testing import example_data
>>> subjects_dir = '.'
>>> data_dir = '.'
>>> output_dir = '.'
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> group_id = 'group1'
>>> l2pipeline = groupwork.create_merge_networks_by_group_workflow(group_list,
↳ group_id, data_dir, subjects_dir, output_dir)
>>> l2pipeline.run()
```

Inputs:

```
group_list: Dictionary of subject lists, keyed by group name
group_id: String containing the group name
data_dir: Path to the data directory
subjects_dir: Path to the Freesurfer 'subjects' directory
output_dir: Path for the output files
```

pullnodeIDs ()

[Link to code](#)

This function will return the values contained, for each node in a network, given an input key. By default it will return the node names

workflows.dmri.connectivity.nx

create_cmats_to_csv_pipeline()

[Link to code](#)

Creates a workflow to convert the outputs from CreateMatrix into a single comma-separated value text file. An extra column / field is also added to the text file. Typically, the user would connect the subject name to this field.

Example

```
>>> from nipy.workflows.dmri.connectivity.nx import create_cmats_to_csv_pipeline
>>> csv = create_cmats_to_csv_pipeline("cmats_to_csv", "subject_id")
>>> csv.inputs.inputnode.extra_field = 'subj1'
>>> csv.inputs.inputnode.matlab_matrix_files = ['subj1_cmatrix.mat', 'subj1_mean_
↪ fiber_length.mat', 'subj1_median_fiber_length.mat', 'subj1_fiber_length_std.mat
↪']
>>> csv.run()
```

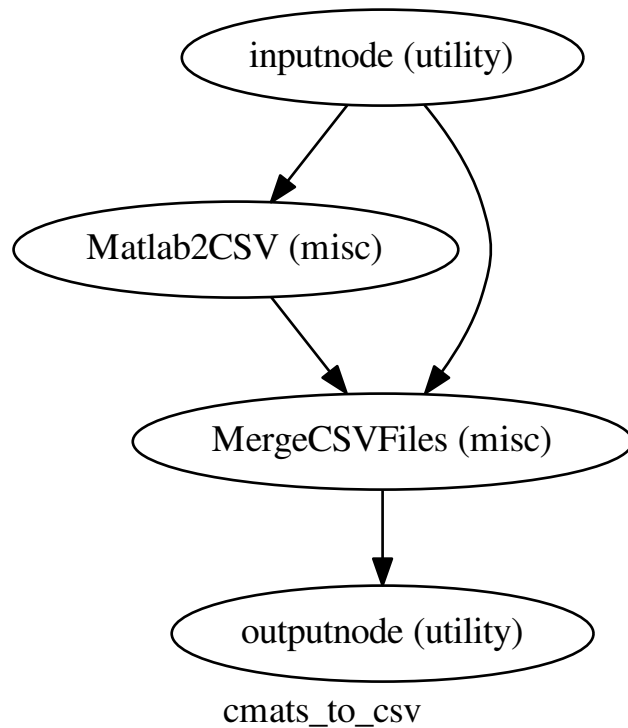
Inputs:

```
inputnode.extra_field
inputnode.matlab_matrix_files
```

Outputs:

```
outputnode.csv_file
```

Graph



create_networkx_pipeline()

[Link to code](#)

Creates a workflow to calculate various graph measures (via NetworkX) on an input network. The output measures are then converted to comma-separated value text files, and an extra column / field is also added. Typically, the user would connect the subject name to this field.

Example

```

>>> from nipy.workflows.dmri.connectivity.nx import create_networkx_pipeline
>>> nx = create_networkx_pipeline("networkx", "subject_id")
>>> nx.inputs.inputnode.extra_field = 'subj1'
>>> nx.inputs.inputnode.network_file = 'subj1.pck'
>>> nx.run()

```

Inputs:

```

inputnode.extra_field
inputnode.network_file

```

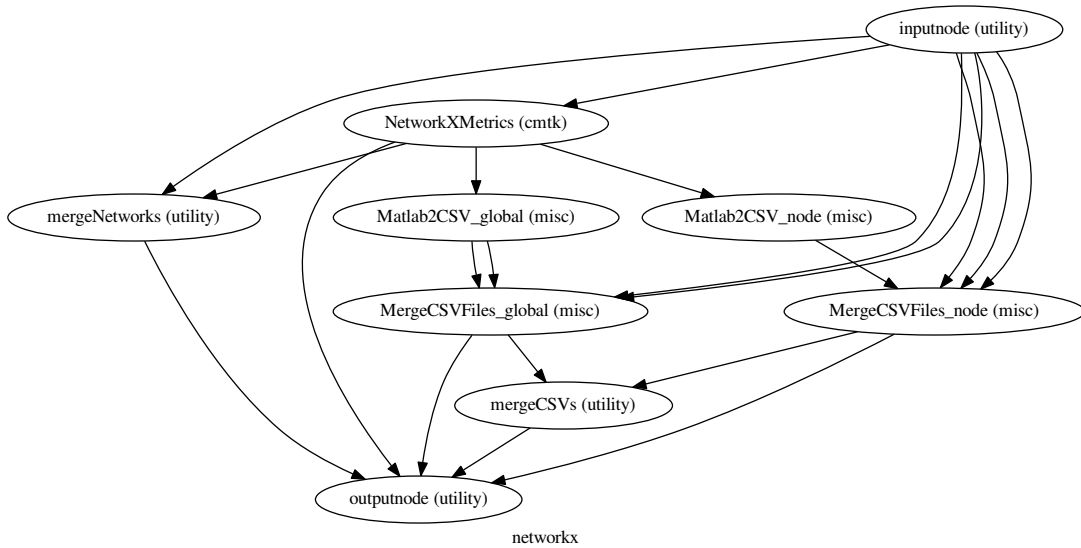
Outputs:

```

outputnode.network_files
outputnode.csv_files
outputnode.matlab_files

```

Graph



add_global_to_filename()

[Link to code](#)

add_nodal_to_filename()

[Link to code](#)

workflows.dmri.dipy.denoise

nlmeans_pipeline()

[Link to code](#)

Workflow that performs nlmeans denoising

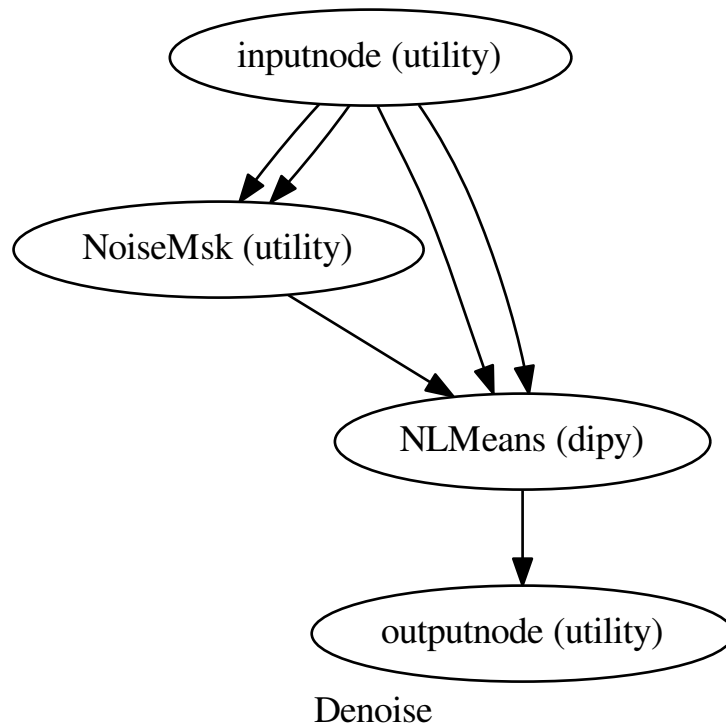
Example

```

>>> from nipy.workflows.dmri.dipy.denoise import nlmeans_pipeline
>>> denoise = nlmeans_pipeline()
>>> denoise.inputs.inputnode.in_file = 'diffusion.nii'
>>> denoise.inputs.inputnode.in_mask = 'mask.nii'
>>> denoise.run()

```

Graph



bg_mask()

[Link to code](#)

Rough mask of background from brain masks

csf_mask()

[Link to code](#)

Artesanal mask of csf in T2w-like images

workflows.dmri.fsl.artifacts

all_fmb_pipeline()

[Link to code](#)

Builds a pipeline including three artifact corrections: head-motion correction (HMC), susceptibility-derived distortion correction (SDC), and Eddy currents-derived distortion correction (ECC).

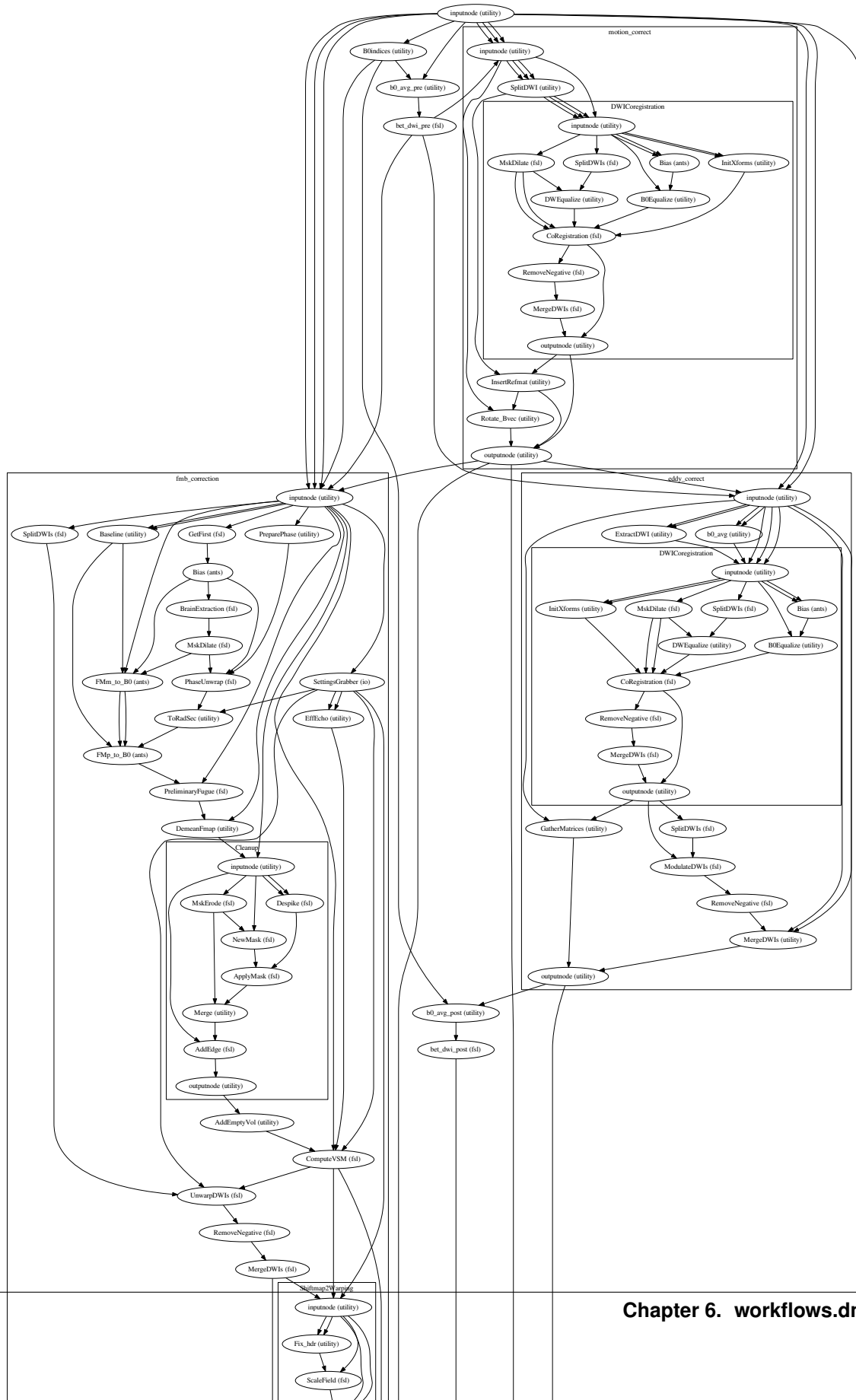
The displacement fields from each kind of distortions are combined. Thus, only one interpolation occurs between input data and result.

Warning: this workflow rotates the gradients table (*b*-vectors) [*Leemans09*].

Examples

```
>>> from nipy.workflows.dmri.fsl.artifacts import all_fmb_pipeline
>>> allcorr = all_fmb_pipeline()
>>> allcorr.inputs.inputnode.in_file = 'epi.nii'
>>> allcorr.inputs.inputnode.in_bval = 'diffusion.bval'
>>> allcorr.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> allcorr.inputs.inputnode.bmap_mag = 'magnitude.nii'
>>> allcorr.inputs.inputnode.bmap pha = 'phase.nii'
>>> allcorr.inputs.inputnode.epi_param = 'epi_param.txt'
>>> allcorr.run()
```

Graph



`all_fsl_pipeline()`

[Link to code](#)

Workflow that integrates FSL `topup` and `eddy`.

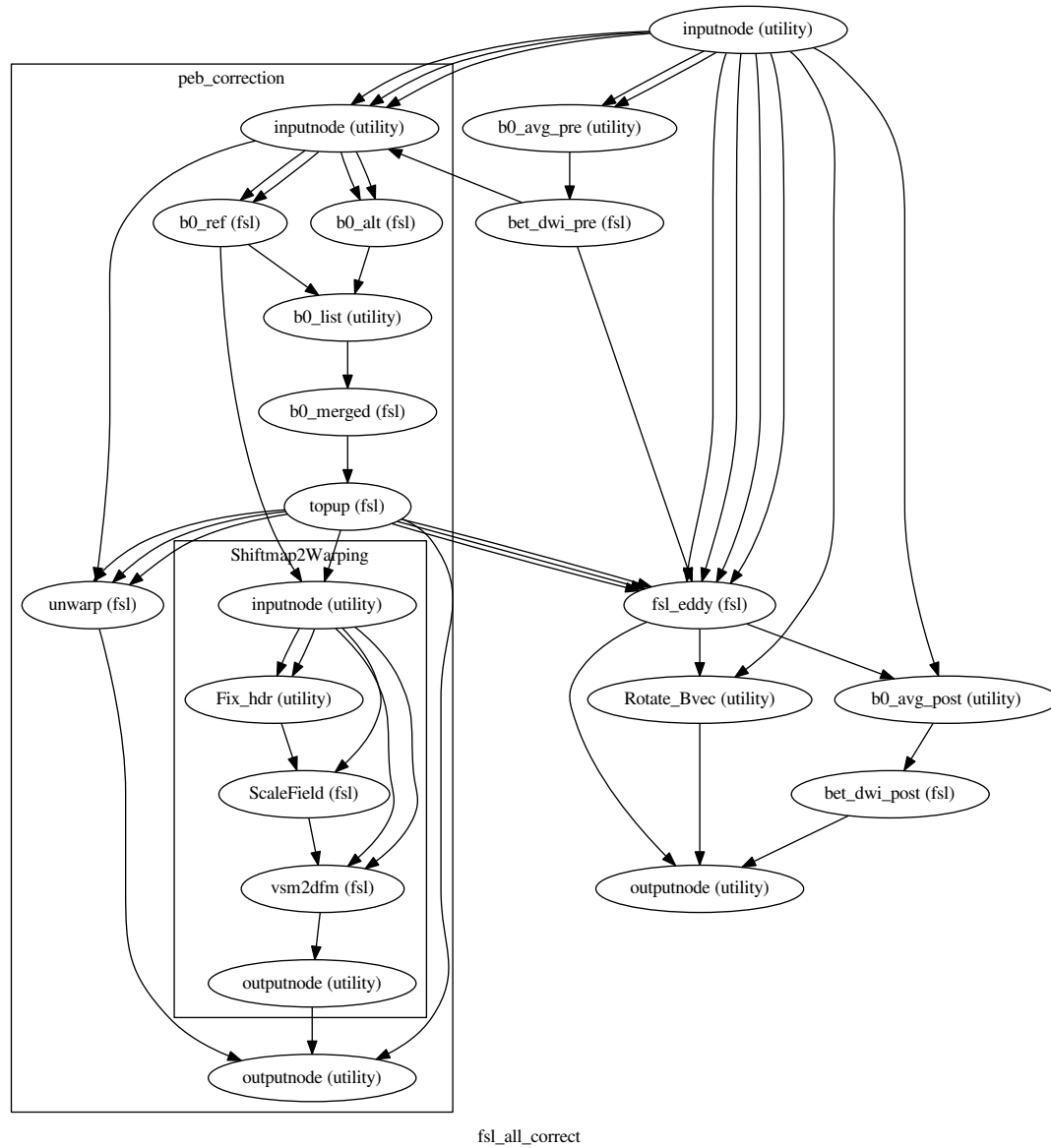
Warning: this workflow rotates the gradients table (*b*-vectors) [*Leemans09*].

Warning: this workflow does not perform jacobian modulation of each *DWI* [*Jones10*].

Examples

```
>>> from nipyre.workflows.dmri.fsl.artifacts import all_fsl_pipeline
>>> allcorr = all_fsl_pipeline()
>>> allcorr.inputs.inputnode.in_file = 'epi.nii'
>>> allcorr.inputs.inputnode.alt_file = 'epi_rev.nii'
>>> allcorr.inputs.inputnode.in_bval = 'diffusion.bval'
>>> allcorr.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> allcorr.run()
```

Graph

`all_peb_pipeline()`[Link to code](#)

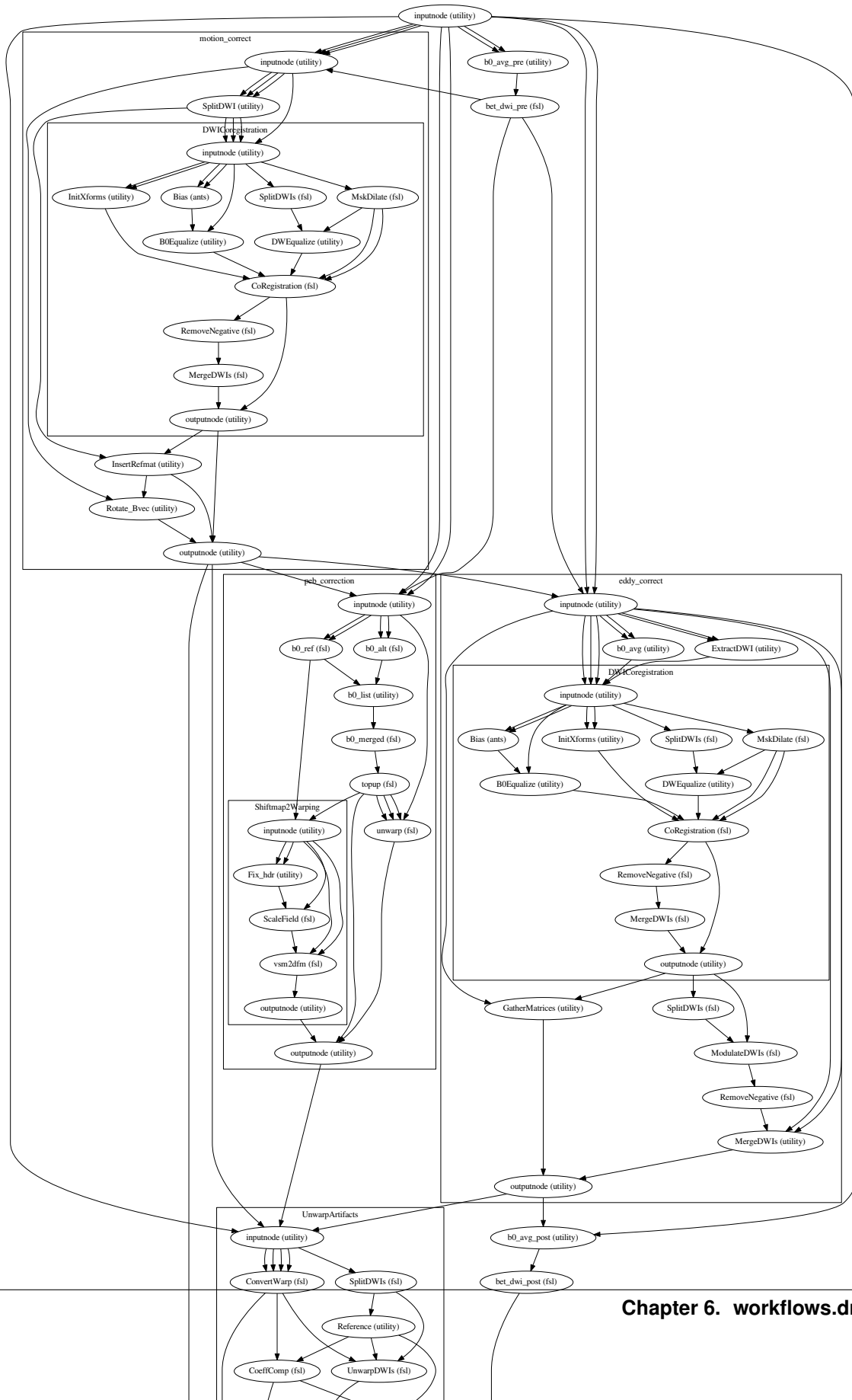
Builds a pipeline including three artifact corrections: head-motion correction (HMC), susceptibility-derived distortion correction (SDC), and Eddy currents-derived distortion correction (ECC).

Warning: this workflow rotates the gradients table (*b*-vectors) [Leemans09].

Examples

```
>>> from nipyype.workflows.dmri.fsl.artifacts import all_peb_pipeline
>>> allcorr = all_peb_pipeline()
>>> allcorr.inputs.inputnode.in_file = 'epi.nii'
>>> allcorr.inputs.inputnode.alt_file = 'epi_rev.nii'
>>> allcorr.inputs.inputnode.in_bval = 'diffusion.bval'
>>> allcorr.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> allcorr.run()
```

Graph



`ecc_pipeline()`

[Link to code](#)

ECC stands for Eddy currents correction.

Creates a pipeline that corrects for artifacts induced by Eddy currents in dMRI sequences. It takes a series of diffusion weighted images and linearly co-registers them to one reference image (the average of all b0s in the dataset).

DWIs are also modulated by the determinant of the Jacobian as indicated by [\[Jones10\]](#) and [\[Rohde04\]](#).

A list of rigid transformation matrices can be provided, sourcing from a `hmc_pipeline()` workflow, to initialize registrations in a *motion free* framework.

A list of affine transformation matrices is available as output, so that transforms can be chained ([discussion here](#)).

References

Example

```
>>> from nipy.workflows.dmri.fsl.artifacts import ecc_pipeline
>>> ecc = ecc_pipeline()
>>> ecc.inputs.inputnode.in_file = 'diffusion.nii'
>>> ecc.inputs.inputnode.in_bval = 'diffusion.bval'
>>> ecc.inputs.inputnode.in_mask = 'mask.nii'
>>> ecc.run()
```

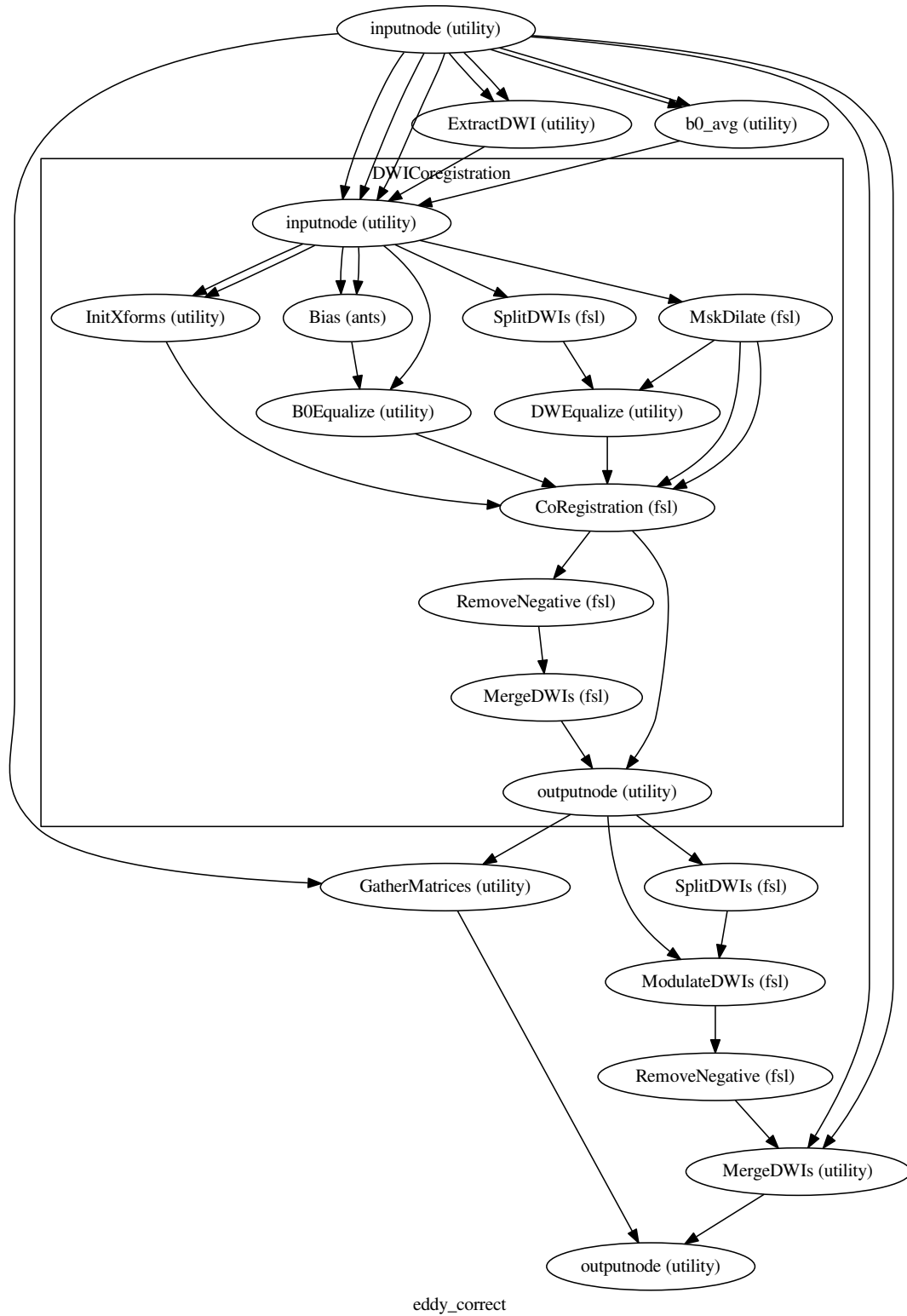
Inputs:

```
inputnode.in_file - input dwi file
inputnode.in_mask - weights mask of reference image (a file with data range sin_
↳ [0.0, 1.0], indicating the weight of each voxel when computing the metric.
inputnode.in_bval - b-values table
inputnode.in_xfms - list of matrices to initialize registration (from head-motion_
↳ correction)
```

Outputs:

```
outputnode.out_file - corrected dwi file
outputnode.out_xfms - list of transformation matrices
```

Graph



`hmc_pipeline()`

[Link to code](#)

HMC stands for head-motion correction.

Creates a pipeline that corrects for head motion artifacts in dMRI sequences. It takes a series of diffusion weighted images and rigidly co-registers them to one reference image. Finally, the *b*-matrix is rotated accordingly [\[Leemans09\]](#) making use of the rotation matrix obtained by FLIRT.

Search angles have been limited to 4 degrees, based on results in [\[Yendiki13\]](#).

A list of rigid transformation matrices is provided, so that transforms can be chained. This is useful to correct for artifacts with only one interpolation process (as previously discussed [here](#)), and also to compute nuisance regressors as proposed by [\[Yendiki13\]](#).

Warning: This workflow rotates the *b*-vectors, so please be advised that not all the dicom converters ensure the consistency between the resulting nifti orientation and the gradients table (e.g. dcm2nii checks it).

References

Example

```
>>> from nipy.workflows.dmri.fsl.artifacts import hmc_pipeline
>>> hmc = hmc_pipeline()
>>> hmc.inputs.inputnode.in_file = 'diffusion.nii'
>>> hmc.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> hmc.inputs.inputnode.in_bval = 'diffusion.bval'
>>> hmc.inputs.inputnode.in_mask = 'mask.nii'
>>> hmc.run()
```

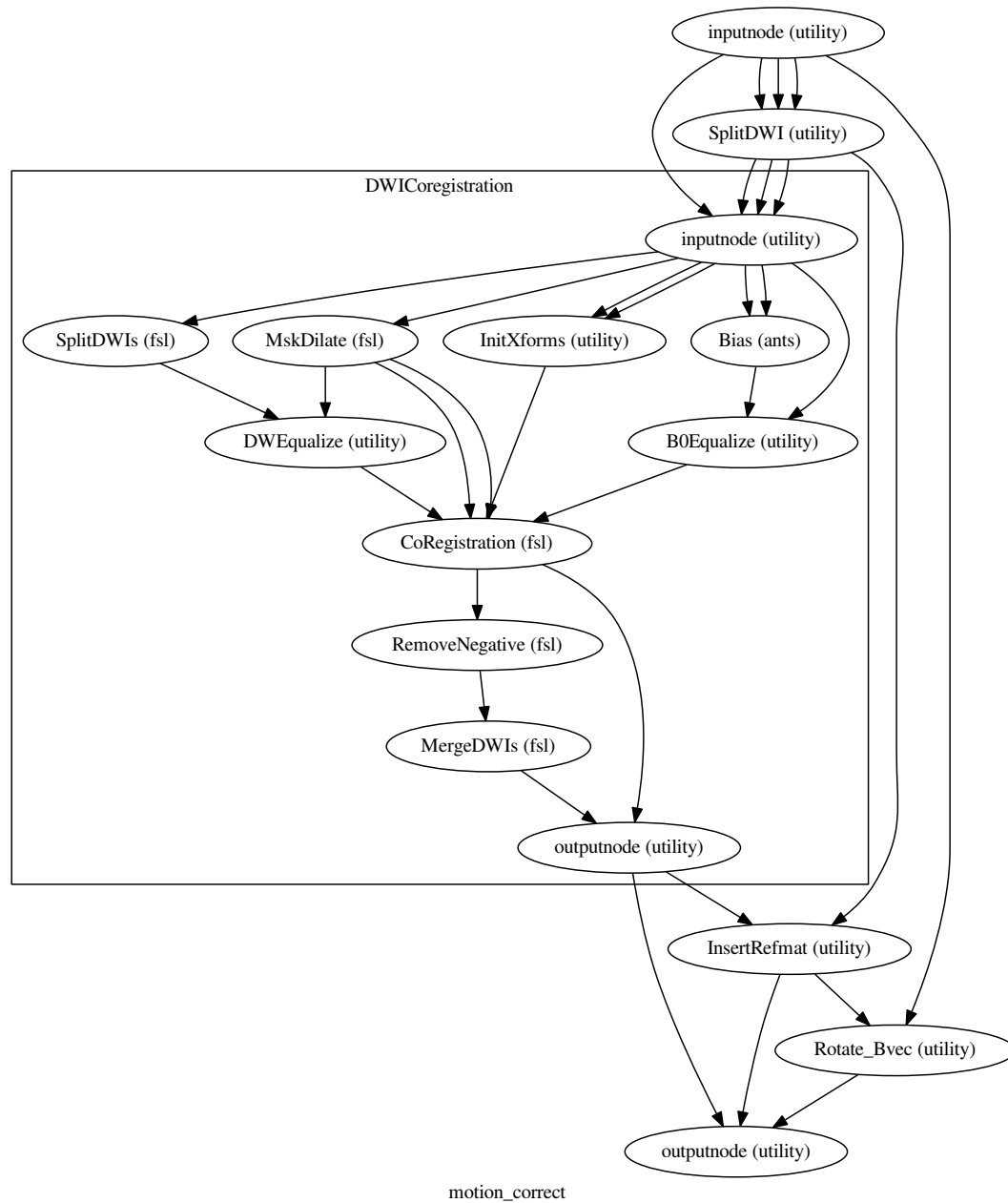
Inputs:

```
inputnode.in_file - input dwi file
inputnode.in_mask - weights mask of reference image (a file with data range in [0.
→0, 1.0], indicating the weight of each voxel when computing the metric.
inputnode.in_bval - b-values file
inputnode.in_bvec - gradients file (b-vectors)
inputnode.ref_num (optional, default=0) index of the b0 volume that should be
→taken as reference
```

Outputs:

```
outputnode.out_file - corrected dwi file
outputnode.out_bvec - rotated gradient vectors table
outputnode.out_xfms - list of transformation matrices
```

Graph

**remove_bias()**

[Link to code](#)

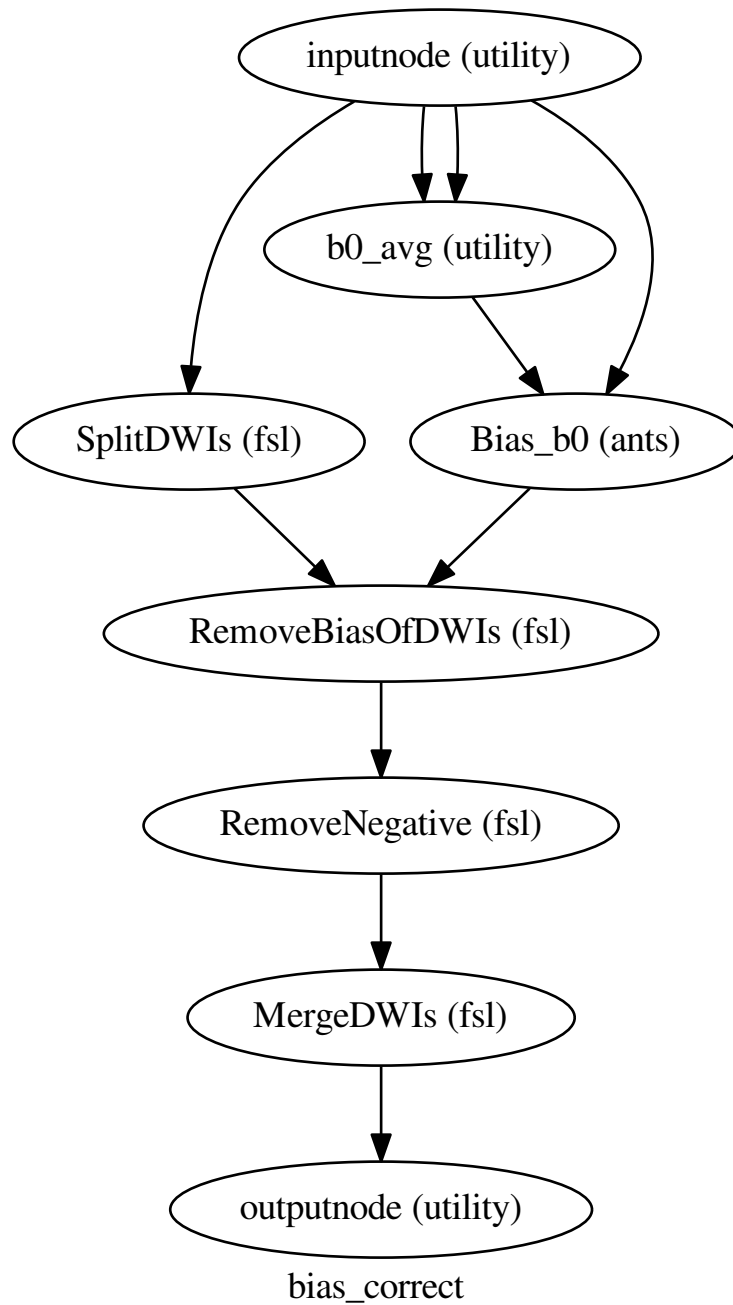
This workflow estimates a single multiplicative bias field from the averaged *b0* image, as suggested in [\[Jeurissen2014\]](#).

References

Example

```
>>> from nipyype.workflows.dmri.fsl.artifacts import remove_bias
>>> bias = remove_bias()
>>> bias.inputs.inputnode.in_file = 'epi.nii'
>>> bias.inputs.inputnode.in_bval = 'diffusion.bval'
>>> bias.inputs.inputnode.in_mask = 'mask.nii'
>>> bias.run()
```

Graph

**`sdc_fmb()`**[Link to code](#)

SDC stands for susceptibility distortion correction. FMB stands for fieldmap-based.

The fieldmap based (FMB) method implements SDC by using a mapping of the B0 field as proposed by [Jezzard95]. This workflow uses the implementation of FSL (FUGUE). Phase unwrapping is performed using PRELUDE [Jenkinson03]. Preparation of the fieldmap is performed reproducing the script in FSL `fsl_prepare_fieldmap`.

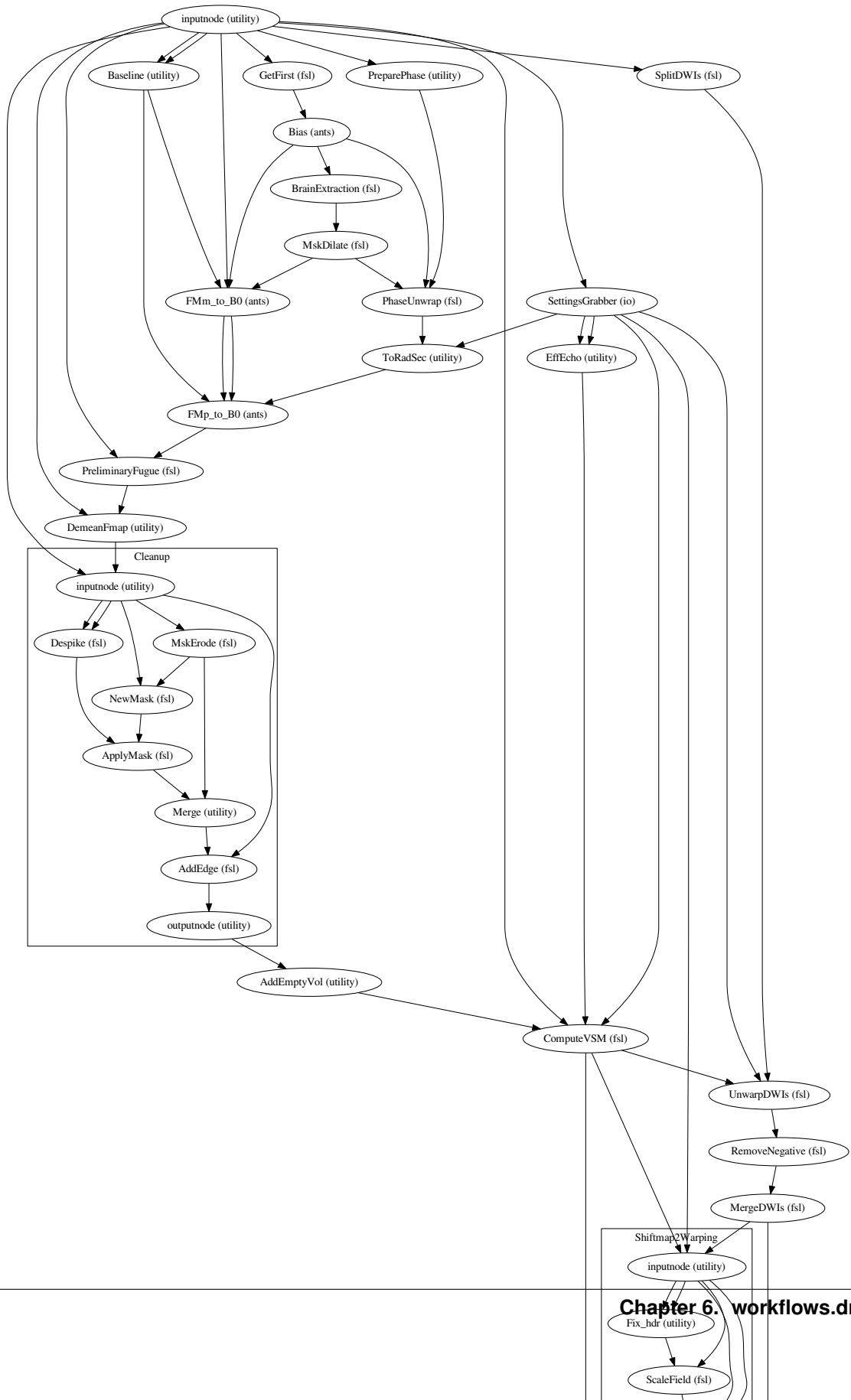
Example

```
>>> from nipy.workflows.dmri.fsl.artifacts import sdc_fmb
>>> fmb = sdc_fmb()
>>> fmb.inputs.inputnode.in_file = 'diffusion.nii'
>>> fmb.inputs.inputnode.in_ref = list(range(0, 30, 6))
>>> fmb.inputs.inputnode.in_mask = 'mask.nii'
>>> fmb.inputs.inputnode.bmap_mag = 'magnitude.nii'
>>> fmb.inputs.inputnode.bmap pha = 'phase.nii'
>>> fmb.inputs.inputnode.settings = 'epi_param.txt'
>>> fmb.run()
```

Warning: Only SIEMENS format fieldmaps are supported.

References

Graph



sd­c_­peb()

[Link to code](#)

SDC stands for susceptibility distortion correction. PEB stands for phase-encoding-based.

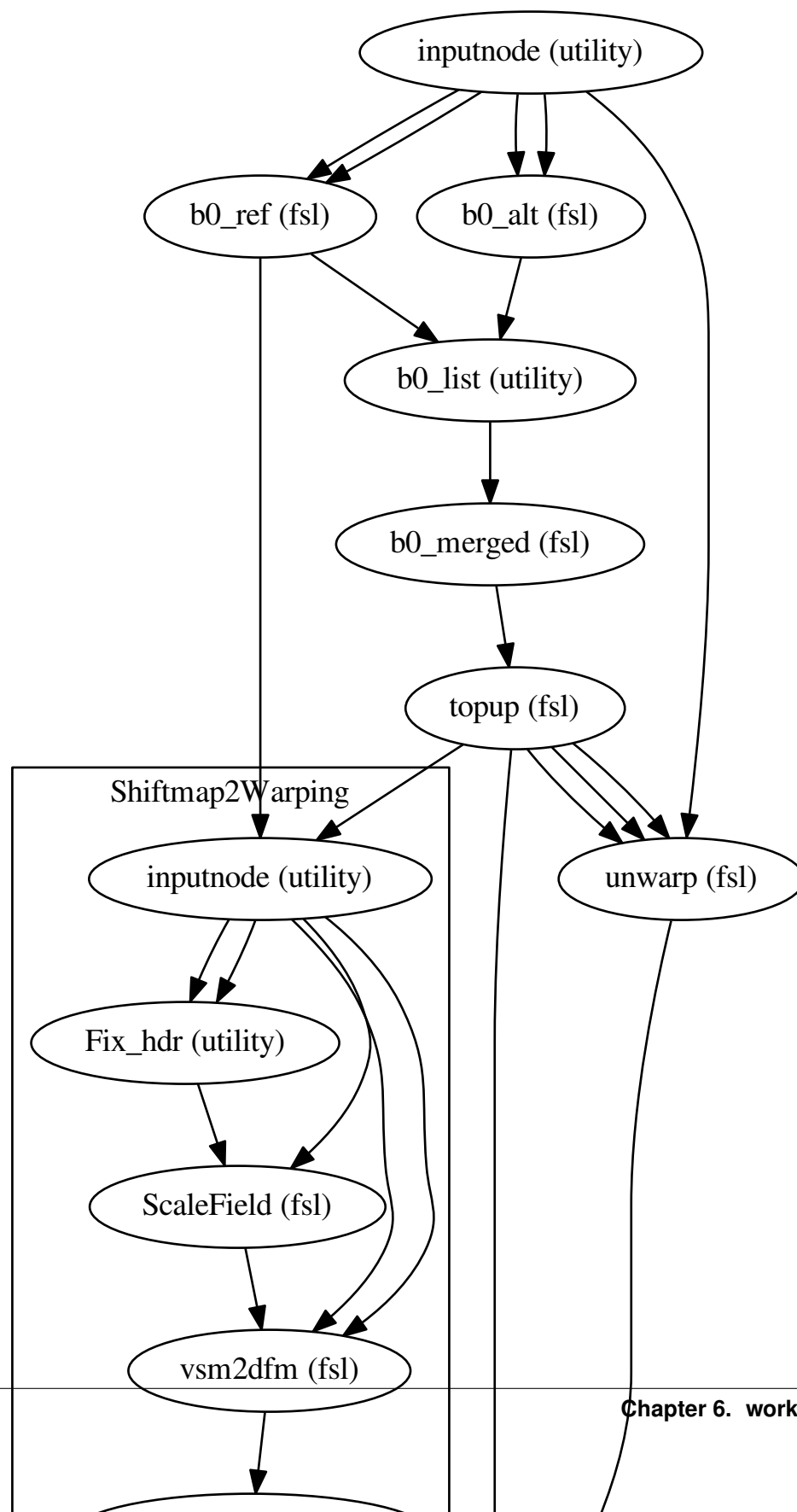
The phase-encoding-based (PEB) method implements SDC by acquiring diffusion images with two different encoding directions [Andersson2003]. The most typical case is acquiring with opposed phase-gradient blips (e.g. $A \ggg P$ and $P \ggg A$, or equivalently, $-y$ and y) as in [Chiou2000], but it is also possible to use orthogonal configurations [Cordes2000] (e.g. $A \ggg P$ and $L \ggg R$, or equivalently $-y$ and x). This workflow uses the implementation of FSL (TOPUP).

Example

```
>>> from nipy.workflows.dmri.fsl.artifacts import sdc_peb
>>> peb = sdc_peb()
>>> peb.inputs.inputnode.in_file = 'epi.nii'
>>> peb.inputs.inputnode.alt_file = 'epi_rev.nii'
>>> peb.inputs.inputnode.in_bval = 'diffusion.bval'
>>> peb.inputs.inputnode.in_mask = 'mask.nii'
>>> peb.run()
```

References

Graph



workflows.dmri.fsl.dti

bedpostx_parallel()

[Link to code](#)

Does the same as `create_bedpostx_pipeline()` by splitting the input dMRI in small ROIs that are better suited for parallel processing).

Example

```
>>> from nipyype.workflows.dmri.fsl.dti import bedpostx_parallel
>>> params = dict(n_fibres = 2, fudge = 1, burn_in = 1000,
...               n_jumps = 1250, sample_every = 25)
>>> bpwf = bedpostx_parallel('nipyype_bedpostx_parallel', params=params)
>>> bpwf.inputs.inputnode.dwi = 'diffusion.nii'
>>> bpwf.inputs.inputnode.mask = 'mask.nii'
>>> bpwf.inputs.inputnode.bvecs = 'bvecs'
>>> bpwf.inputs.inputnode.bvals = 'bvals'
>>> bpwf.run(plugin='CondorDAGMan')
```

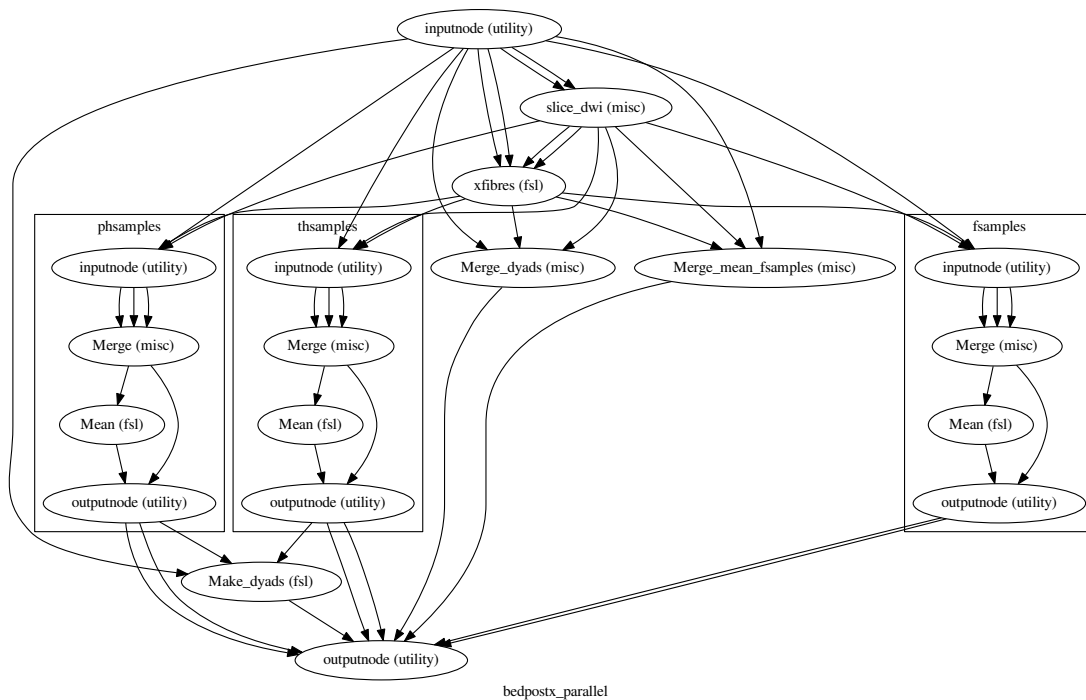
Inputs:

```
inputnode.dwi
inputnode.mask
inputnode.bvecs
inputnode.bvals
```

Outputs:

```
outputnode wraps all XFibres outputs
```

Graph



create_bedpostx_pipeline()

[Link to code](#)

Creates a pipeline that does the same as bedpostx script from FSL - calculates diffusion model parameters (distributions not MLE) voxelwise for the whole volume (by splitting it slicewise).

Example

```
>>> from nipy.workflows.dmri.fsl.dti import create_bedpostx_pipeline
>>> params = dict(n_fibres = 2, fudge = 1, burn_in = 1000,
...               n_jumps = 1250, sample_every = 25)
>>> bpwf = create_bedpostx_pipeline('nipy_bedpostx', params)
>>> bpwf.inputs.inputnode.dwi = 'diffusion.nii'
>>> bpwf.inputs.inputnode.mask = 'mask.nii'
>>> bpwf.inputs.inputnode.bvecs = 'bvecs'
>>> bpwf.inputs.inputnode.bvals = 'bvals'
>>> bpwf.run()
```

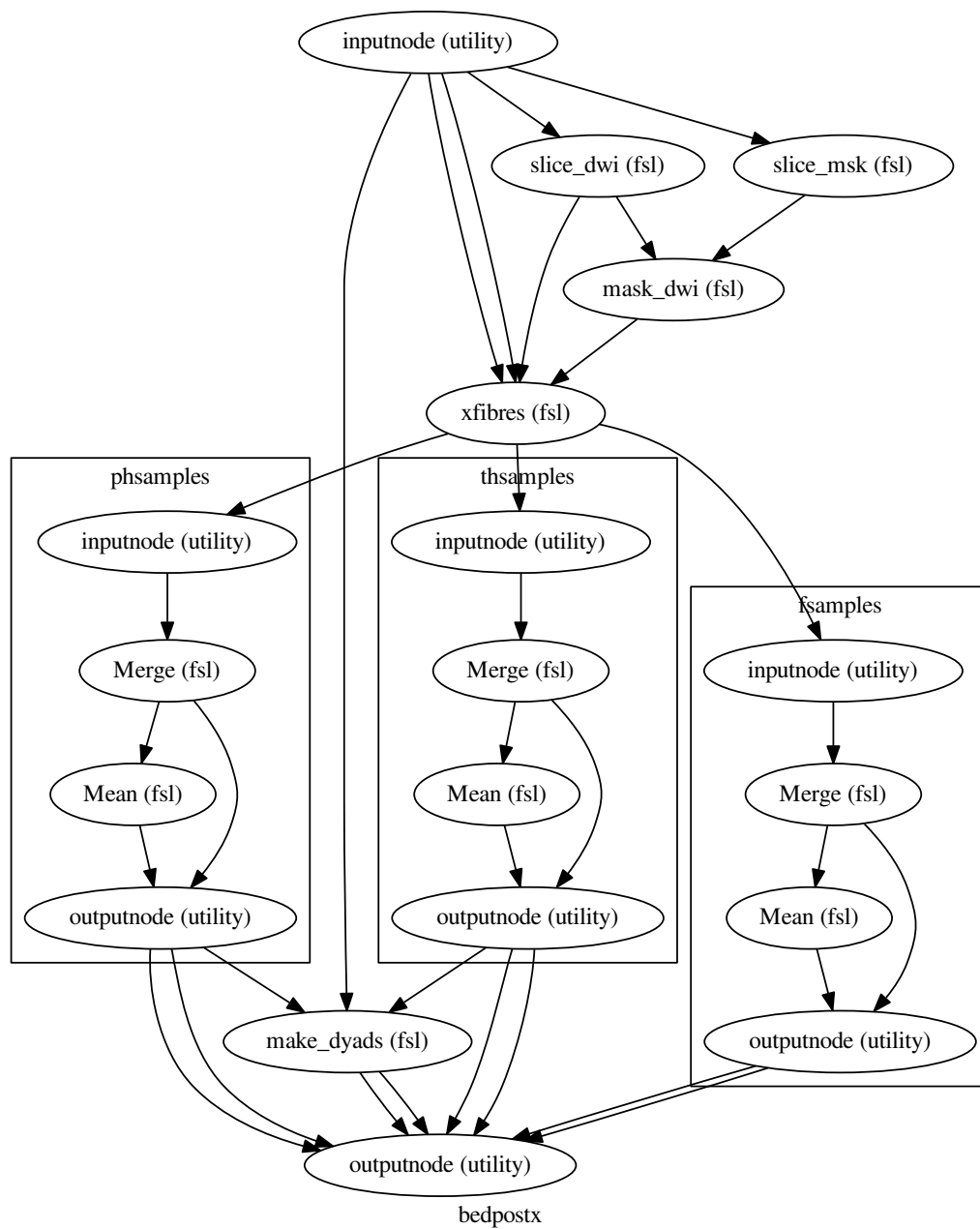
Inputs:

```
inputnode.dwi
inputnode.mask
inputnode.bvecs
inputnode.bvals
```

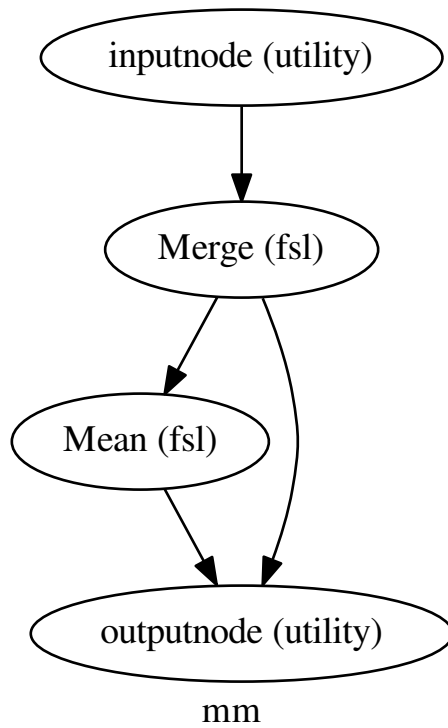
Outputs:

```
outputnode wraps all XFibres outputs
```

Graph

`merge_and_mean()`[Link to code](#)

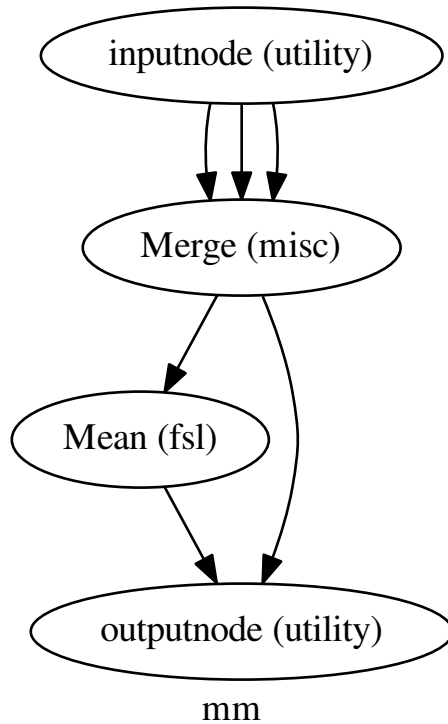
Graph



`merge_and_mean_parallel()`

[Link to code](#)

Graph



`transpose()`

[Link to code](#)

`workflows.dmri.fsl.epi`

`create_dmri_preprocessing()`

[Link to code](#)

Creates a workflow that chains the necessary pipelines to correct for motion, eddy currents, and, if selected, susceptibility artifacts in EPI dMRI sequences.

Deprecated since version 0.9.3: Use `nipyte.workflows.dmri.preprocess.epi.all_fmb_pipeline()` or `nipyte.workflows.dmri.preprocess.epi.all_peb_pipeline()` instead.

Warning: This workflow rotates the b-vectors, so please be advised that not all the dicom converters ensure the consistency between the resulting nifti orientation and the b matrix table (e.g. `dcm2nii` checks it).

Example

```
>>> nipyre_dmri_preprocess = create_dmri_preprocessing('nipyre_dmri_prep')
>>> nipyre_dmri_preprocess.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipyre_dmri_preprocess.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> nipyre_dmri_preprocess.inputs.inputnode.ref_num = 0
>>> nipyre_dmri_preprocess.inputs.inputnode.fieldmap_mag = 'magnitude.nii'
>>> nipyre_dmri_preprocess.inputs.inputnode.fieldmap pha = 'phase.nii'
>>> nipyre_dmri_preprocess.inputs.inputnode.te_diff = 2.46
>>> nipyre_dmri_preprocess.inputs.inputnode.epi_echospacing = 0.77
>>> nipyre_dmri_preprocess.inputs.inputnode.epi_rev_encoding = False
>>> nipyre_dmri_preprocess.inputs.inputnode.pi_accel_factor = True
>>> nipyre_dmri_preprocess.run()
```

Inputs:

```
inputnode.in_file - The diffusion data
inputnode.in_bvec - The b-matrix file, in FSL format and consistent with the in_
↳file orientation
inputnode.ref_num - The reference volume (a b=0 volume in dMRI)
inputnode.fieldmap_mag - The magnitude of the fieldmap
inputnode.fieldmap pha - The phase difference of the fieldmap
inputnode.te_diff - TE increment used (in msec.) on the fieldmap acquisition_
↳(generally 2.46ms for 3T scanners)
inputnode.epi_echospacing - The EPI EchoSpacing parameter (in msec.)
inputnode.epi_rev_encoding - True if reverse encoding was used (generally False)
inputnode.pi_accel_factor - Parallel imaging factor (aka GRAPPA acceleration_
↳factor)
inputnode.vsm_sigma - Sigma (in mm.) of the gaussian kernel used for in-slice_
↳smoothing of the deformation field (voxel shift map, vsm)
```

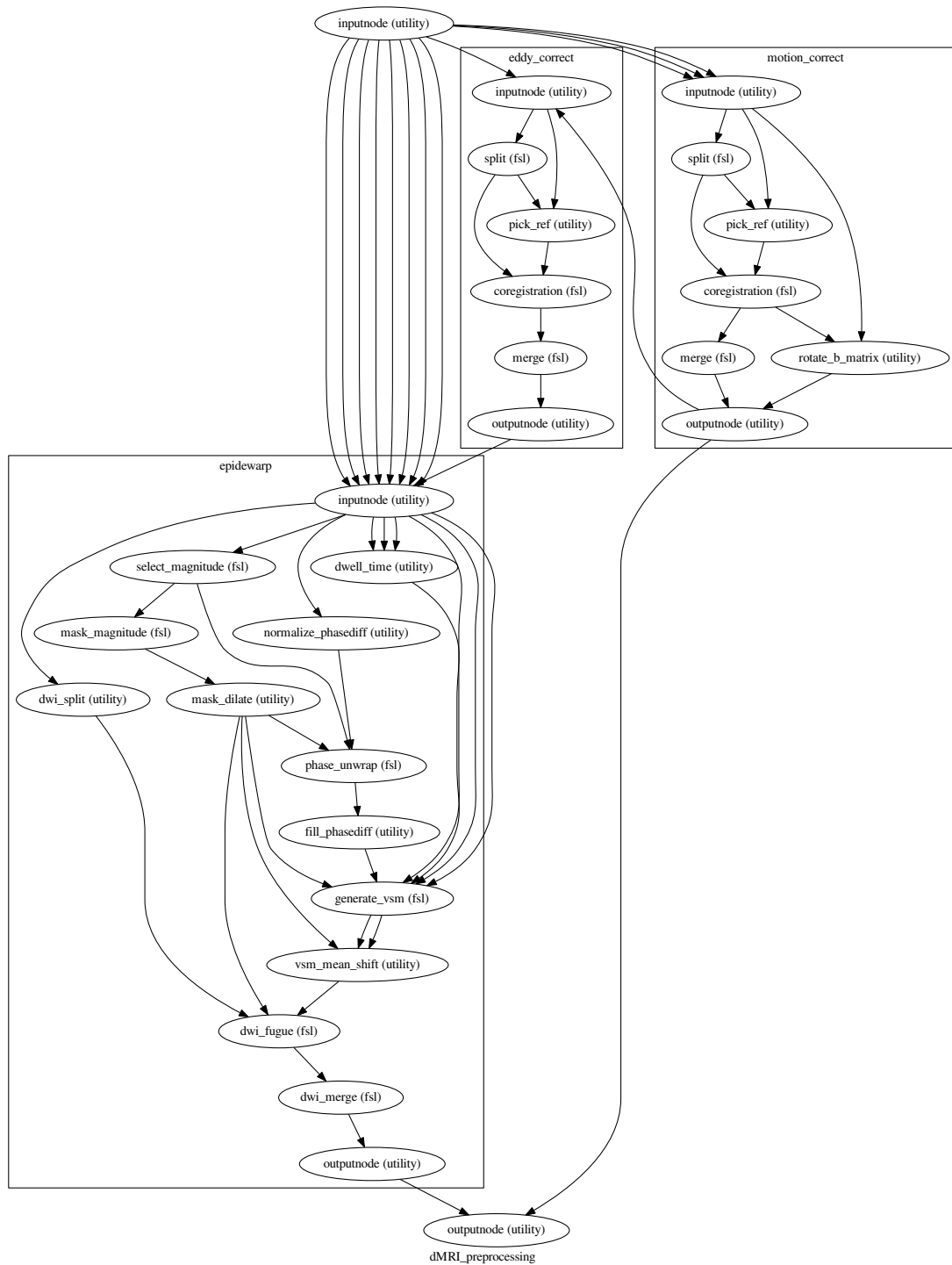
Outputs:

```
outputnode.dmri_corrected
outputnode.bvec_rotated
```

Optional arguments:

```
use_fieldmap - True if there are fieldmap files that should be used (default True)
fieldmap_registration - True if registration to fieldmap should be performed_
↳(default False)
```

Graph



`create_eddy_correct_pipeline()`

[Link to code](#)

Deprecated since version 0.9.3: Use `nipype.workflows.dmri.preprocess.epi.ecc_pipeline()` instead.

Creates a pipeline that replaces `eddy_correct` script in FSL. It takes a series of diffusion weighted images and linearly co-registers them to one reference image. No rotation of the B-matrix is performed, so this pipeline should be executed after the motion correction pipeline.

Example

```
>>> nipype_eddycorrect = create_eddy_correct_pipeline('nipype_eddycorrect')
>>> nipype_eddycorrect.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipype_eddycorrect.inputs.inputnode.ref_num = 0
>>> nipype_eddycorrect.run()
```

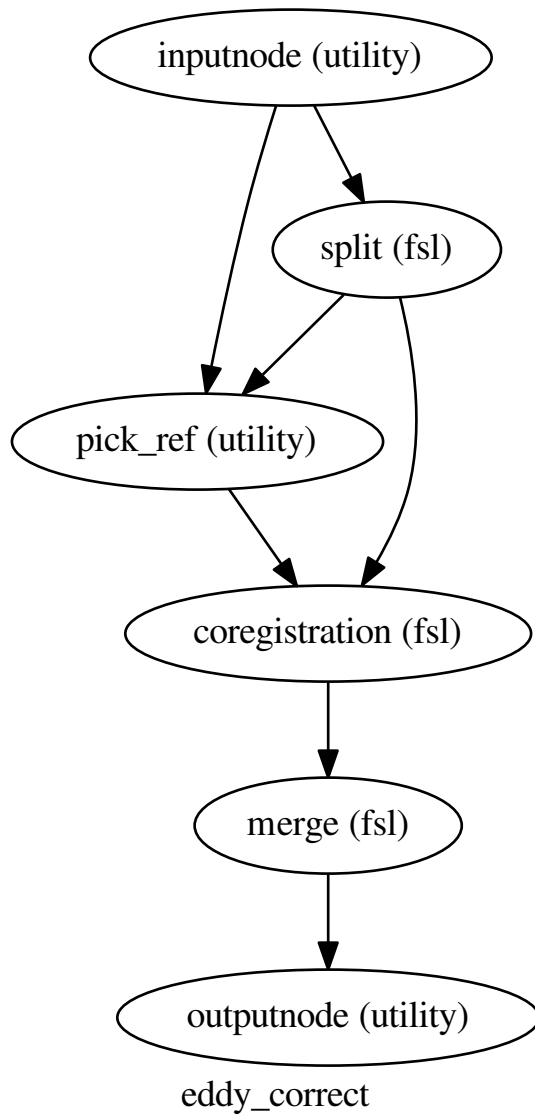
Inputs:

```
inputnode.in_file
inputnode.ref_num
```

Outputs:

```
outputnode.eddy_corrected
```

Graph



`eddy_correct`

`create_epidewarp_pipeline()`

[Link to code](#)

Replaces the `epidewarp.fsl` script (<http://www.nmr.mgh.harvard.edu/~greve/fbirn/b0/epidewarp.fsl>) for susceptibility distortion correction of dMRI & fMRI acquired with EPI sequences and the fieldmap information (Jezzard et al., 1995) using FSL's FUGUE. The registration to the (warped) fieldmap (strictly following the original script) is available using `fieldmap_registration=True`.

Warning: This workflow makes use of `epidewarp.fsl` a script of FSL deprecated long time ago. The use of this workflow is not recommended, use `nipyne.workflows.dmri.preprocess.epi.sdc_fmb()` instead.

Example

```
>>> nipyne_epicorrect = create_epidewarp_pipeline('nipyne_epidewarp', fieldmap_
↳registration=False)
>>> nipyne_epicorrect.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipyne_epicorrect.inputs.inputnode.fieldmap_mag = 'magnitude.nii'
>>> nipyne_epicorrect.inputs.inputnode.fieldmap pha = 'phase.nii'
>>> nipyne_epicorrect.inputs.inputnode.te_diff = 2.46
>>> nipyne_epicorrect.inputs.inputnode.epi_echospacing = 0.77
>>> nipyne_epicorrect.inputs.inputnode.epi_rev_encoding = False
>>> nipyne_epicorrect.inputs.inputnode.ref_num = 0
>>> nipyne_epicorrect.inputs.inputnode.pi_accel_factor = 1.0
>>> nipyne_epicorrect.run()
```

Inputs:

`inputnode.in_file` - The volume acquired **with** EPI sequence
`inputnode.fieldmap_mag` - The magnitude of the fieldmap
`inputnode.fieldmap pha` - The phase difference of the fieldmap
`inputnode.te_diff` - Time difference between TE **in** ms.
`inputnode.epi_echospacing` - The echo spacing (aka dwell time) **in** the EPI sequence
`inputnode.epi_ph_encoding_dir` - The phase encoding direction **in** EPI acquisition_
↳ (default y)
`inputnode.epi_rev_encoding` - **True if** it **is** acquired **with** reverse encoding
`inputnode.pi_accel_factor` - Acceleration factor used **for** EPI parallel imaging_
↳ (GRAPPA)
`inputnode.vsm_sigma` - Sigma value of the gaussian smoothing **filter** applied to the_
↳ vsm (voxel shift **map**)
`inputnode.ref_num` - The reference volume (B=0 **in** dMRI **or** a central frame **in** fMRI)

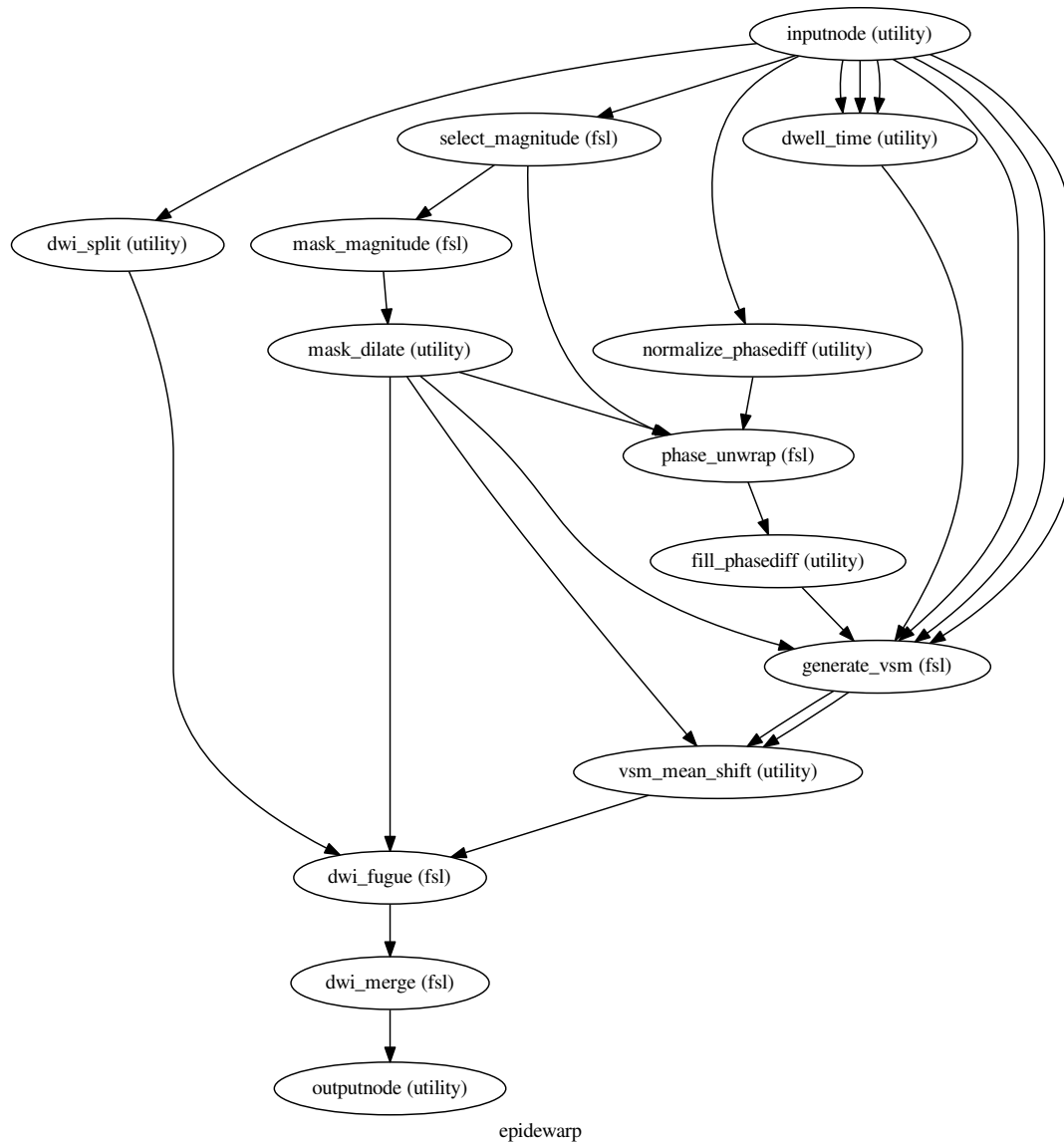
Outputs:

```
outputnode.epi_corrected
```

Optional arguments:

`fieldmap_registration` - **True if** registration to fieldmap should be done (default_
↳ **False**)

Graph

**create_motion_correct_pipeline()**

[Link to code](#)

Creates a pipeline that corrects for motion artifact in dMRI sequences. It takes a series of diffusion weighted images and rigidly co-registers them to one reference image. Finally, the b-matrix is rotated accordingly (Lee-mans et al. 2009 - <http://www.ncbi.nlm.nih.gov/pubmed/19319973>), making use of the rotation matrix obtained by FLIRT.

Deprecated since version 0.9.3: Use `nipy.workflow.dmri.preprocess.epi.hmc_pipeline()` instead.

Warning: This workflow rotates the b-vectors, so please be advised that not all the dicom converters ensure the consistency between the resulting nifti orientation and the b matrix table (e.g. dcm2nii checks it).

Example

```
>>> nipyne_motioncorrect = create_motion_correct_pipeline('nipyne_motioncorrect')
>>> nipyne_motioncorrect.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipyne_motioncorrect.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> nipyne_motioncorrect.inputs.inputnode.ref_num = 0
>>> nipyne_motioncorrect.run()
```

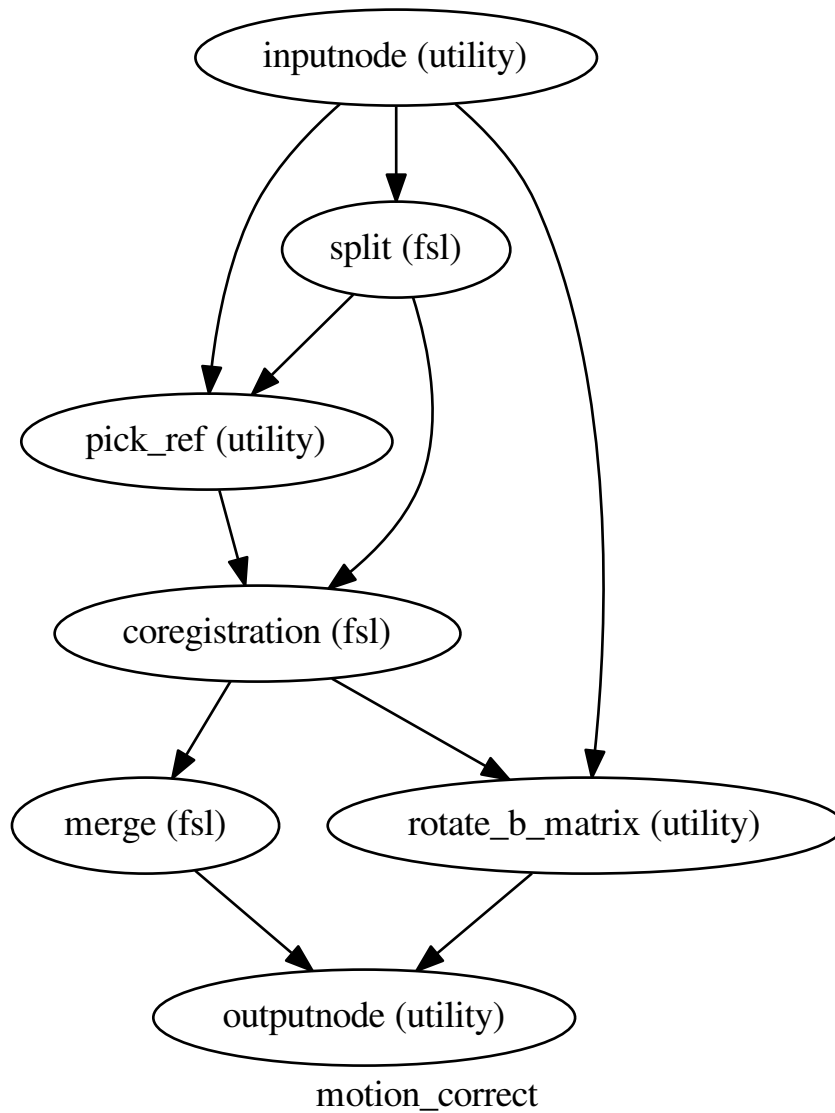
Inputs:

```
inputnode.in_file
inputnode.ref_num
inputnode.in_bvec
```

Outputs:

```
outputnode.motion_corrected
outputnode.out_bvec
```


Graph

**fieldmap_correction()**

[Link to code](#)

Deprecated since version 0.9.3: Use `nipy.workflows.dmri.preprocess.epi.sdc_fmb()` instead.

Fieldmap-based retrospective correction of EPI images for the susceptibility distortion artifact (Jezzard et al., 1995). Fieldmap images are assumed to be already registered to EPI data, and a brain mask is required.

Replaces the former workflow, still available as `create_epidewarp_pipeline()`. The difference with respect the epidewarp pipeline is that now the workflow uses the new `fsl_prepare_fieldmap` available as of FSL 5.0.

Example

```
>>> nipy_epicorrect = fieldmap_correction('nipy_epidewarp')
>>> nipy_epicorrect.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipy_epicorrect.inputs.inputnode.in_mask = 'brainmask.nii'
>>> nipy_epicorrect.inputs.inputnode.fieldmap pha = 'phase.nii'
>>> nipy_epicorrect.inputs.inputnode.fieldmap_mag = 'magnitude.nii'
>>> nipy_epicorrect.inputs.inputnode.te_diff = 2.46
>>> nipy_epicorrect.inputs.inputnode.epi_echospacing = 0.77
>>> nipy_epicorrect.inputs.inputnode.encoding_direction = 'y'
>>> nipy_epicorrect.run()
```

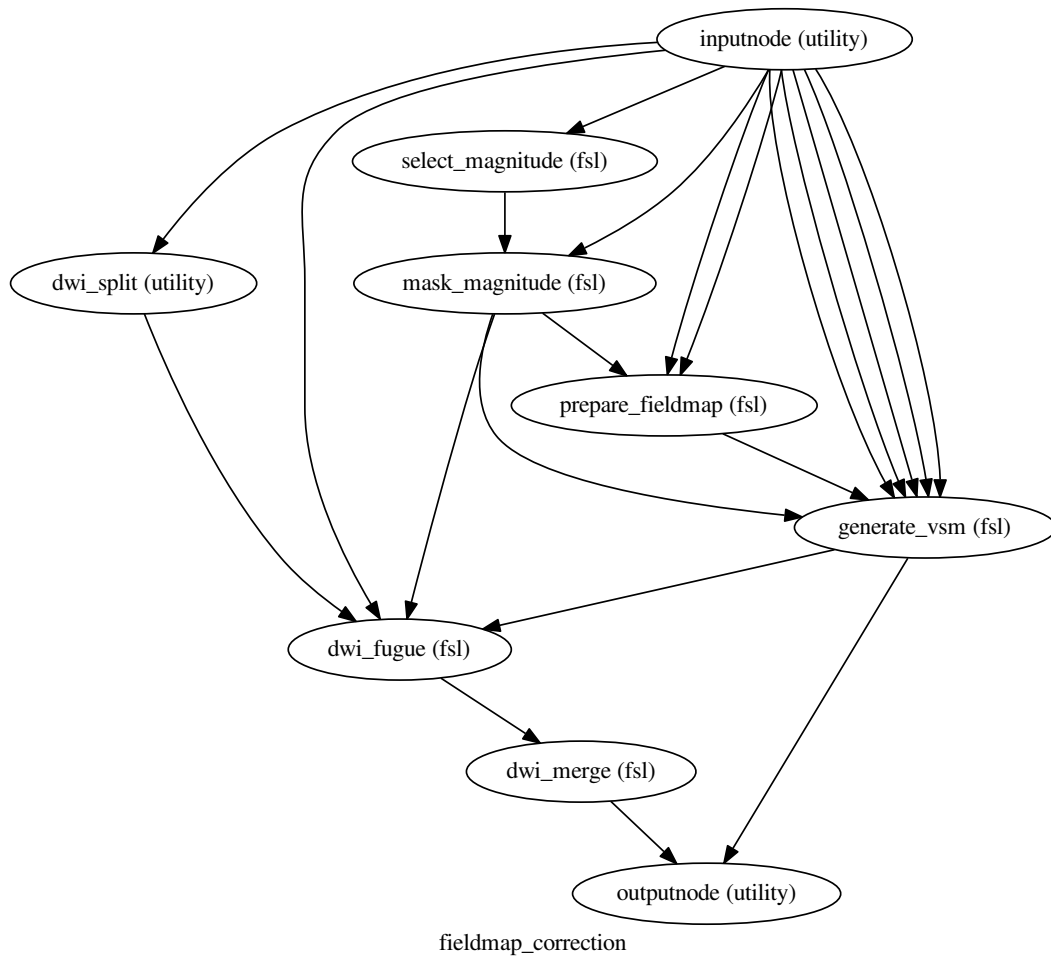
Inputs:

```
inputnode.in_file - The volume acquired with EPI sequence
inputnode.in_mask - A brain mask
inputnode.fieldmap pha - The phase difference map from the fieldmapping,
↳ registered to in_file
inputnode.fieldmap_mag - The magnitud maps (usually 4D, one magnitude per GRE
↳ scan)
                        from the fieldmapping, registered to in_file
inputnode.te_diff - Time difference in msec. between TE in ms of the fieldmapping,
↳ (usually a GRE sequence).
inputnode.epi_echospacing - The effective echo spacing (aka dwell time) in msec.
↳ of the EPI sequence. If
                        EPI was acquired with parallel imaging, then the
↳ effective echo spacing is
                        eff_es = es / acc_factor.
inputnode.encoding_direction - The phase encoding direction in EPI acquisition,
↳ (default y)
inputnode.vsm_sigma - Sigma value of the gaussian smoothing filter applied to the
↳ vsm (voxel shift map)
```

Outputs:

```
outputnode.epi_corrected
outputnode.out_vsm
```

Graph



topup_correction()

[Link to code](#)

Deprecated since version 0.9.3: Use `nipyre.workflows.dmri.preprocess.epi.sdc_peb()` instead.

Corrects for susceptibility distortion of EPI images when one reverse encoding dataset has been acquired

Example

```

>>> nipyre_epicorrect = topup_correction('nipyre_topup')
>>> nipyre_epicorrect.inputs.inputnode.in_file_dir = 'epi.nii'
>>> nipyre_epicorrect.inputs.inputnode.in_file_rev = 'epi_rev.nii'
>>> nipyre_epicorrect.inputs.inputnode.encoding_direction = ['y', 'y-']
>>> nipyre_epicorrect.inputs.inputnode.ref_num = 0
>>> nipyre_epicorrect.run()

```

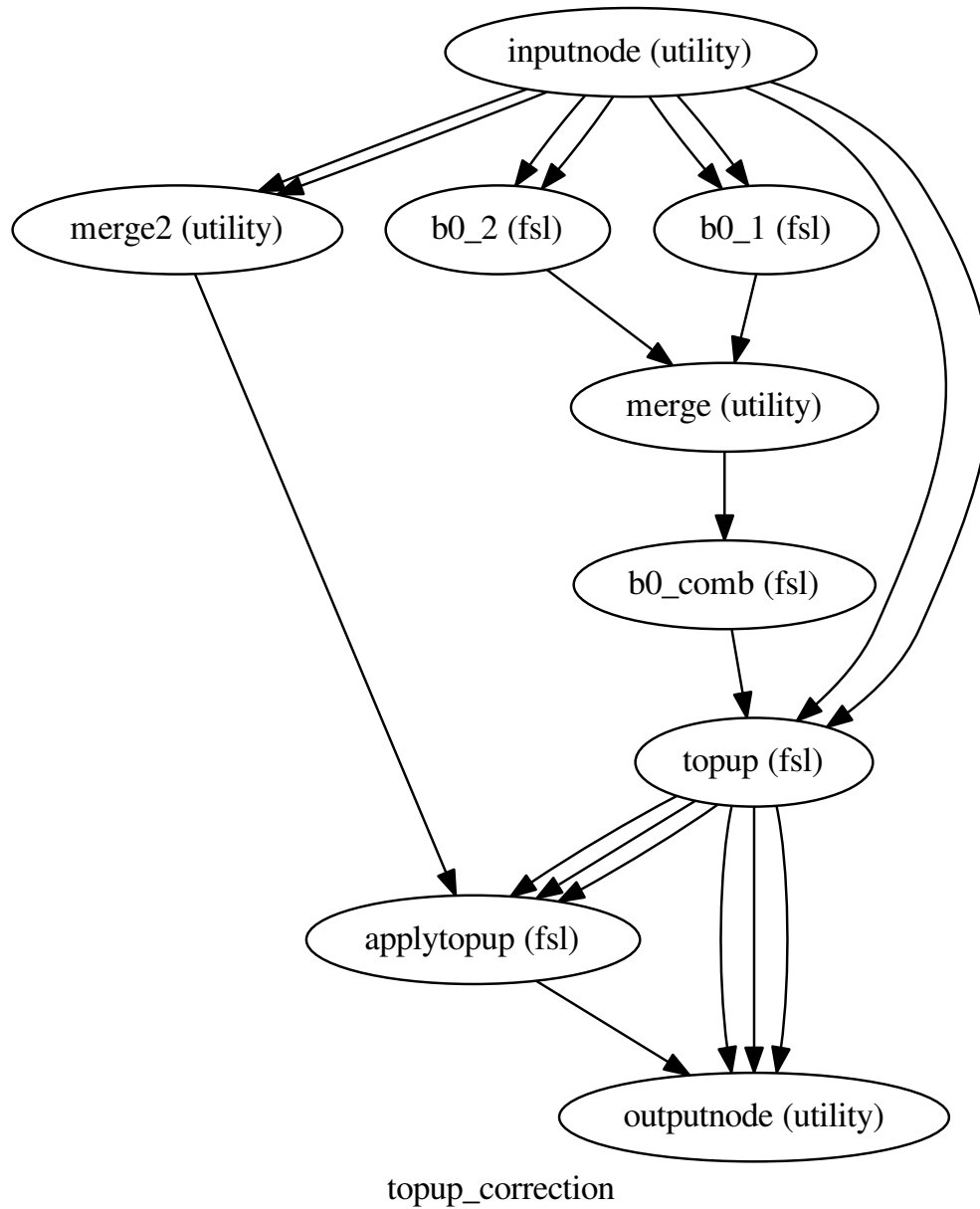
Inputs:

```
inputnode.in_file_dir - EPI volume acquired in 'forward' phase encoding
inputnode.in_file_rev - EPI volume acquired in 'reversed' phase encoding
inputnode.encoding_direction - Direction encoding of in_file_dir
inputnode.ref_num - Identifier of the reference volumes (usually B0 volume)
```

Outputs:

```
outputnode.epi_corrected
```

Graph

**workflows.dmri.fsl.tbss****create_tbss_1_preproc()**[Link to code](#)

Preprocess FA data for TBSS: erodes a little and zero end slicers and creates masks(for use in FLIRT & FNIRT)

from FSL). A pipeline that does the same as tbss_1_preproc script in FSL

Example

```
>>> from nipype.workflows.dmri.fsl import tbss
>>> tbss1 = tbss.create_tbss_1_preproc()
>>> tbss1.inputs.inputnode.fa_list = ['s1_FA.nii', 's2_FA.nii', 's3_FA.nii']
```

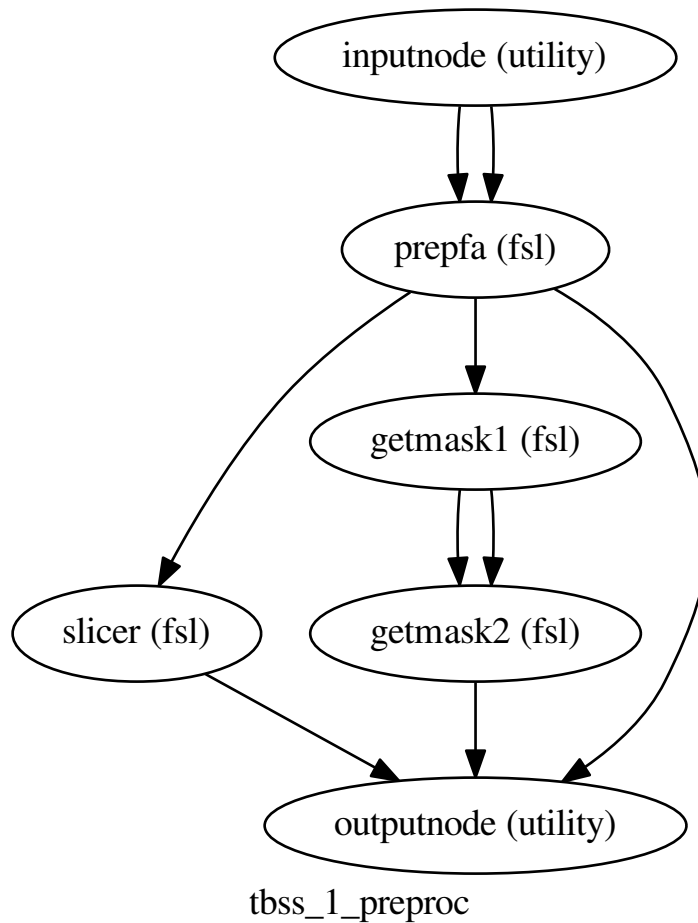
Inputs:

```
inputnode.fa_list
```

Outputs:

```
outputnode.fa_list
outputnode.mask_list
outputnode.slices
```

Graph



create_tbss_2_reg()

[Link to code](#)

TBSS nonlinear registration: A pipeline that does the same as ‘tbss_2_reg -t’ script in FSL. ‘-n’ option is not supported at the moment.

Example

```

>>> from nipyype.workflows.dmri.fsl import tbss
>>> tbss2 = create_tbss_2_reg(name="tbss2")
>>> tbss2.inputs.inputnode.target = fsl.Info.standard_image("FMRIB58_FA_1mm.nii.gz")
>>> tbss2.inputs.inputnode.fa_list = ['s1_FA.nii', 's2_FA.nii', 's3_FA.nii']
>>> tbss2.inputs.inputnode.mask_list = ['s1_mask.nii', 's2_mask.nii', 's3_mask.nii']

```

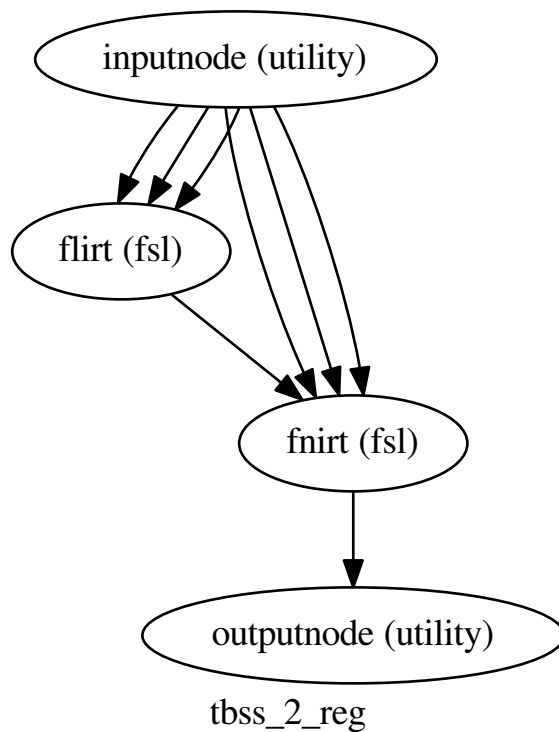
Inputs:

```
inputnode.fa_list  
inputnode.mask_list  
inputnode.target
```

Outputs:

```
outputnode.field_list
```

Graph



`create_tbss_3_postreg()`

[Link to code](#)

Post-registration processing: derive mean_FA and mean_FA_skeleton from mean of all subjects in study. Target is assumed to be FMRIB58_FA_1mm. A pipeline that does the same as 'tbss_3_postreg -S' script from FSL Setting 'estimate_skeleton to False will use precomputed FMRIB58_FA-skeleton_1mm skeleton (same as 'tbss_3_postreg -T').

Example


```
>>> from nipy.workflows.dmri.fsl import tbss
>>> tbss3 = tbss.create_tbss_3_postreg()
>>> tbss3.inputs.inputnode.fa_list = ['s1_wrapped_FA.nii', 's2_wrapped_FA.nii',
↪ 's3_wrapped_FA.nii']
```

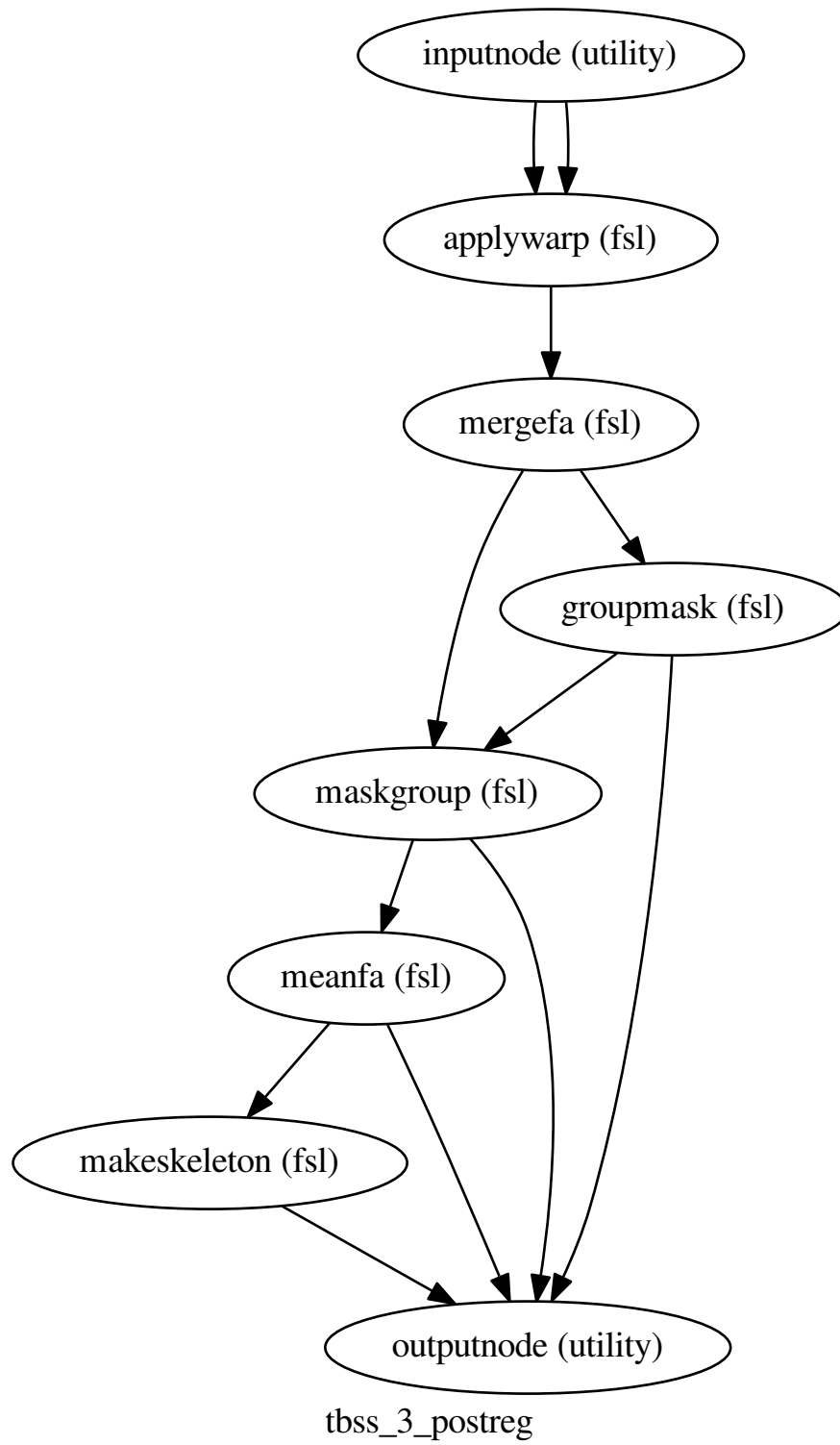
Inputs:

```
inputnode.field_list
inputnode.fa_list
```

Outputs:

```
outputnode.groupmask
outputnode.skeleton_file
outputnode.meanfa_file
outputnode.mergefa_file
```

Graph



`create_tbss_4_prestats()`

[Link to code](#)

Post-registration processing: Creating skeleton mask using a threshold projecting all FA data onto skeleton.

A pipeline that does the same as tbss_4_prestats script from FSL

Example

```
>>> from nipyne.workflows.dmri.fsl import tbss
>>> tbss4 = tbss.create_tbss_4_prestats(name='tbss4')
>>> tbss4.inputs.inputnode.skeleton_thresh = 0.2
```

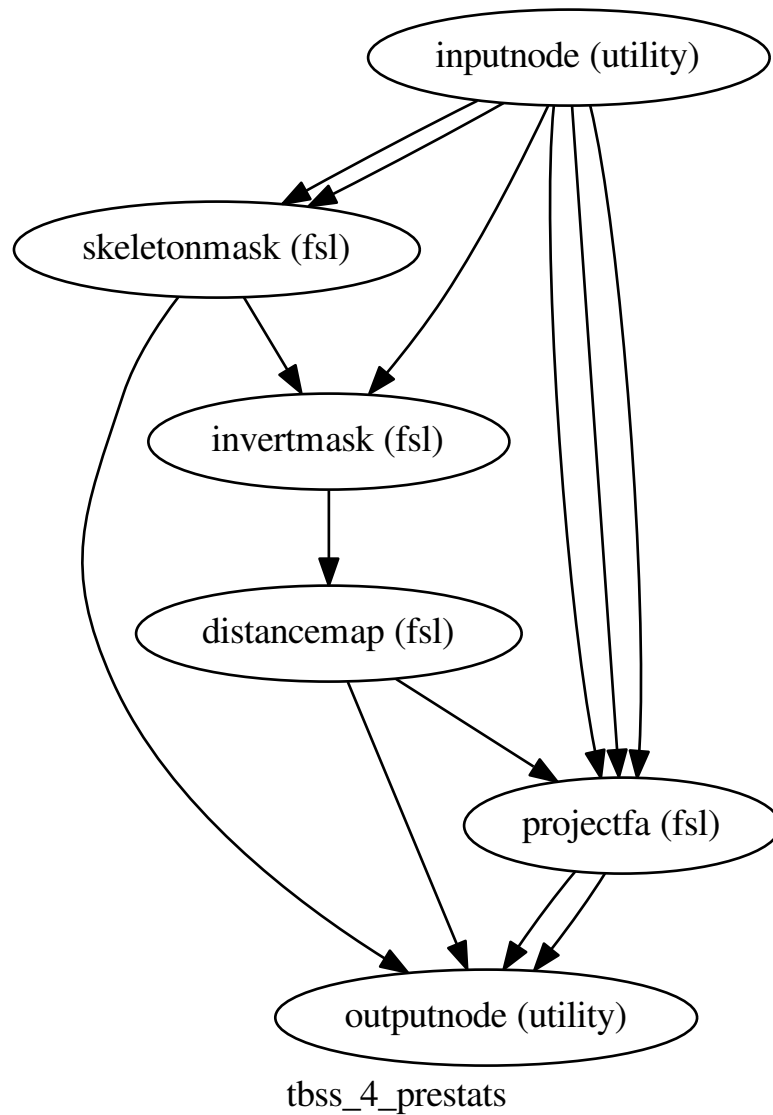
Inputs:

```
inputnode.skeleton_thresh
inputnode.groupmask
inputnode.skeleton_file
inputnode.meanfa_file
inputnode.mergefa_file
```

Outputs:

```
outputnode.all_FA_skeletonised
outputnode.mean_FA_skeleton_mask
outputnode.distance_map
outputnode.skeleton_file
```

Graph



`create_tbss_all()`

[Link to code](#)

Create a pipeline that combines create_tbss_* pipelines

Example

```
>>> from nipyne.workflows.dmri.fsl import tbss
>>> tbss = tbss.create_tbss_all('tbss')
```

```
>>> tbss.inputs.inputnode.skeleton_thresh = 0.2
```

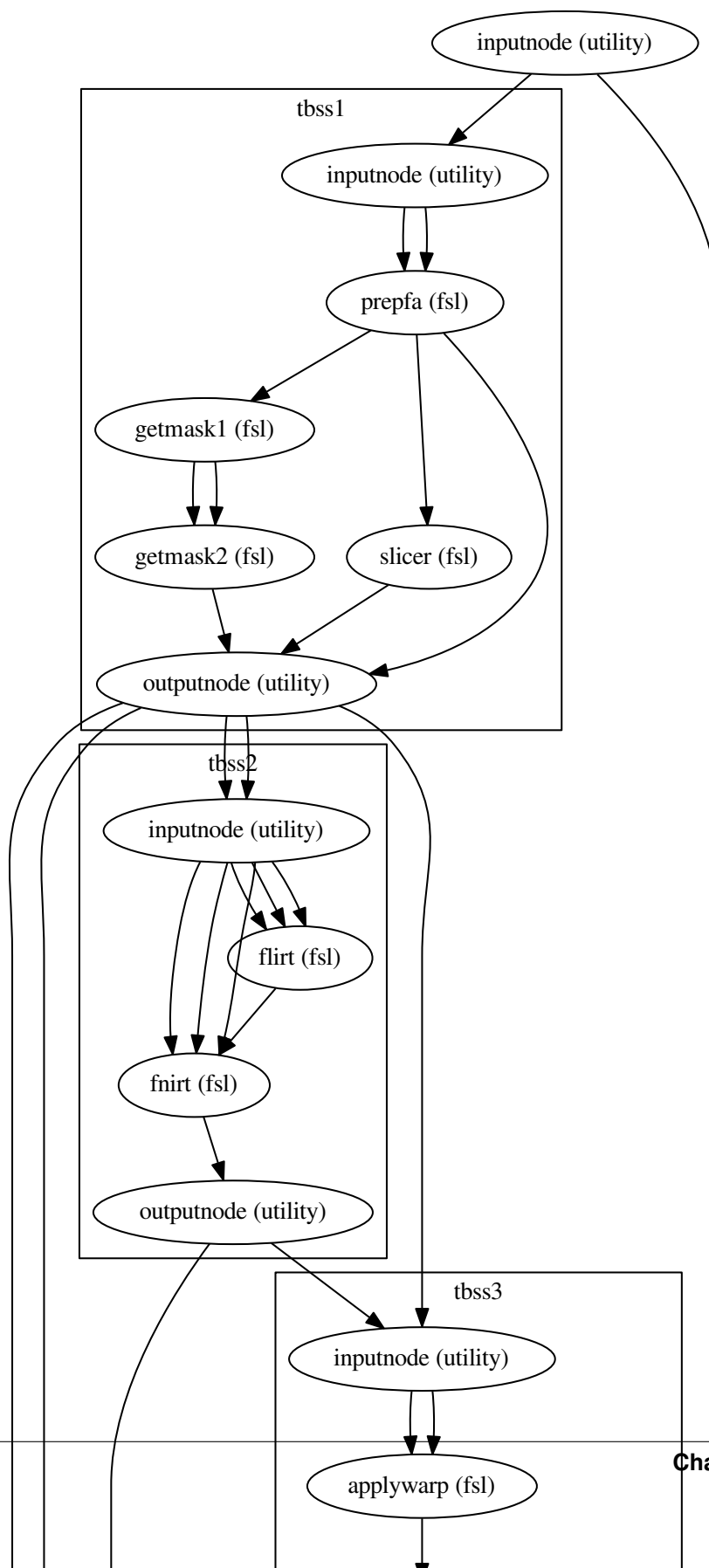
Inputs:

```
inputnode.fa_list  
inputnode.skeleton_thresh
```

Outputs:

```
outputnode.meanfa_file  
outputnode.projectedfa_file  
outputnode.skeleton_file  
outputnode.skeleton_mask
```

Graph



`create_tbss_non_FA()`

[Link to code](#)

A pipeline that implement tbss_non_FA in FSL

Example

```
>>> from nipy.workflows.dmri.fsl import tbss
>>> tbss_MD = tbss.create_tbss_non_FA()
>>> tbss_MD.inputs.inputnode.file_list = []
>>> tbss_MD.inputs.inputnode.field_list = []
>>> tbss_MD.inputs.inputnode.skeleton_thresh = 0.2
>>> tbss_MD.inputs.inputnode.groupmask = './xxx'
>>> tbss_MD.inputs.inputnode.meanfa_file = './xxx'
>>> tbss_MD.inputs.inputnode.distance_map = []
>>> tbss_MD.inputs.inputnode.all_FA_file = './xxx'
```

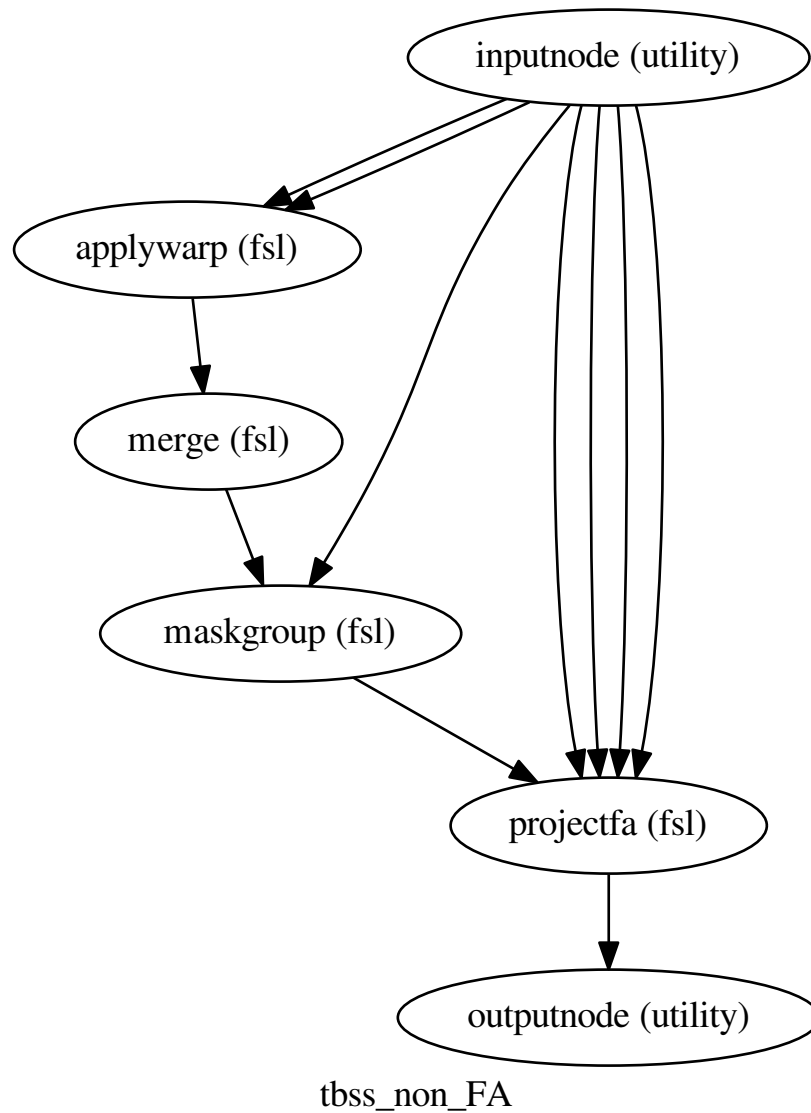
Inputs:

```
inputnode.file_list
inputnode.field_list
inputnode.skeleton_thresh
inputnode.groupmask
inputnode.meanfa_file
inputnode.distance_map
inputnode.all_FA_file
```

Outputs:

```
outputnode.projected_nonFA_file
```

Graph



tbss1_op_string()

[Link to code](#)

tbss4_op_string()

[Link to code](#)

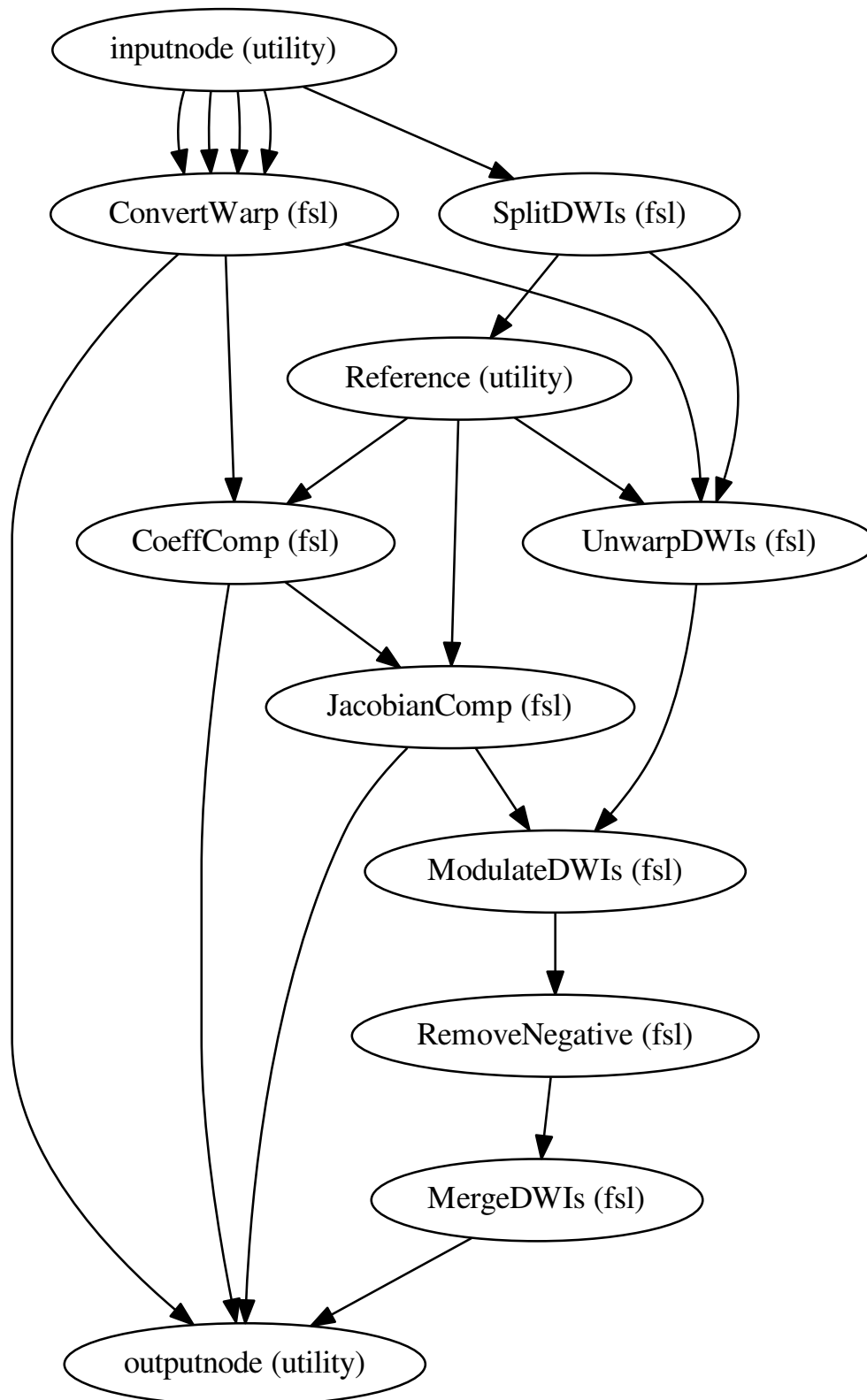
workflows.dmri.fsl.utils

`apply_all_corrections()`

[Link to code](#)

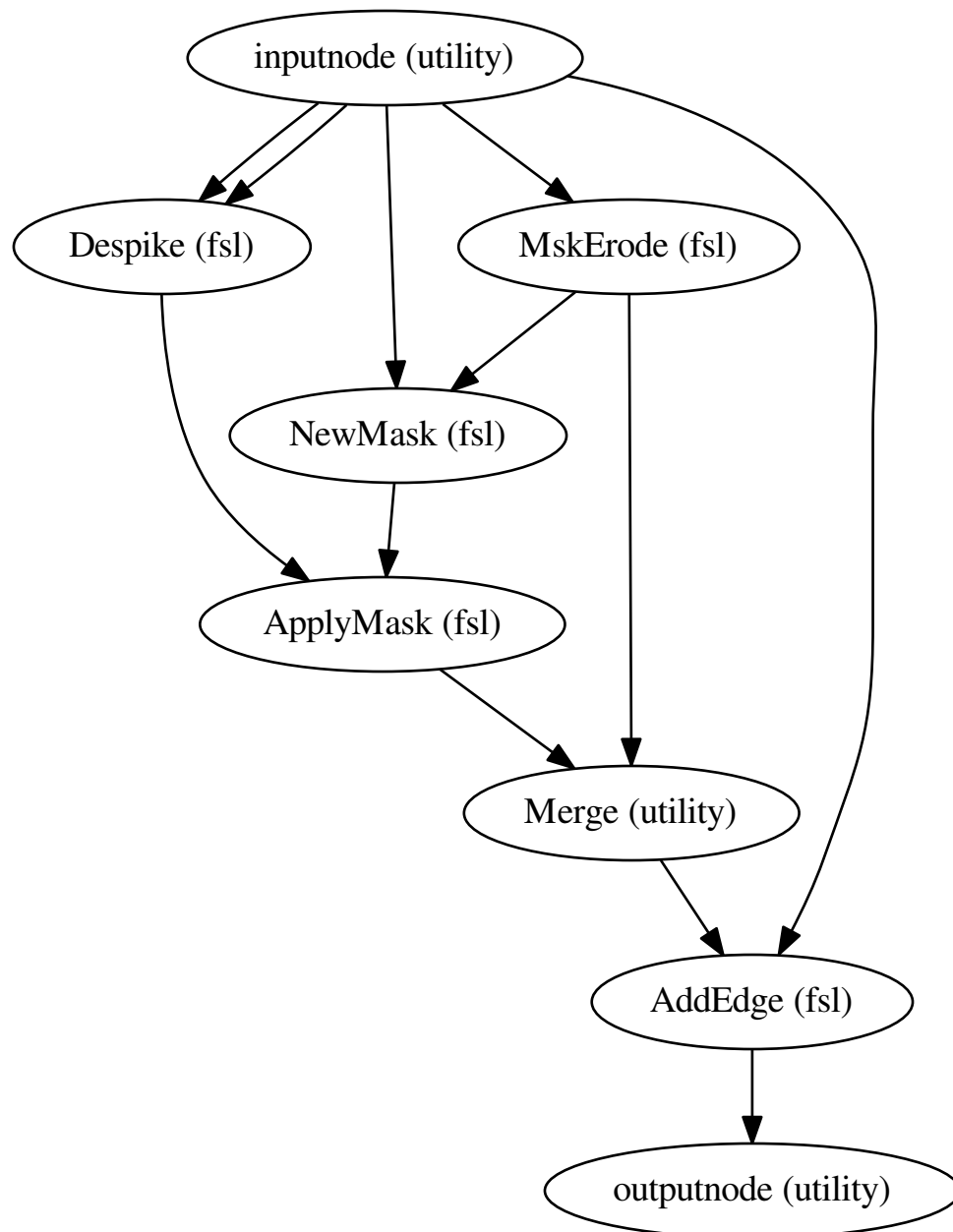
Combines two lists of linear transforms with the deformation field map obtained typically after the SDC process. Additionally, computes the corresponding bspline coefficients and the map of determinants of the jacobian.

Graph



cleanup_edge_pipeline()[Link to code](#)

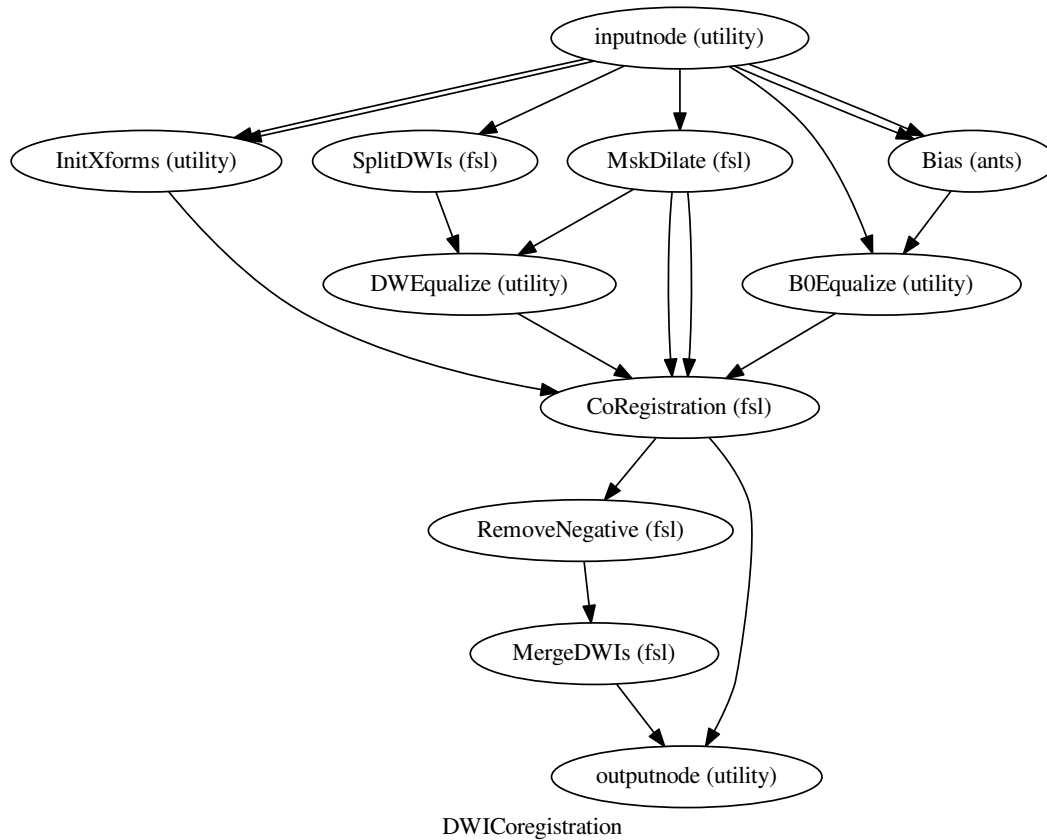
Perform some de-spiking filtering to clean up the edge of the fieldmap (copied from fsl_prepare_fieldmap)

Graph

Cleanup

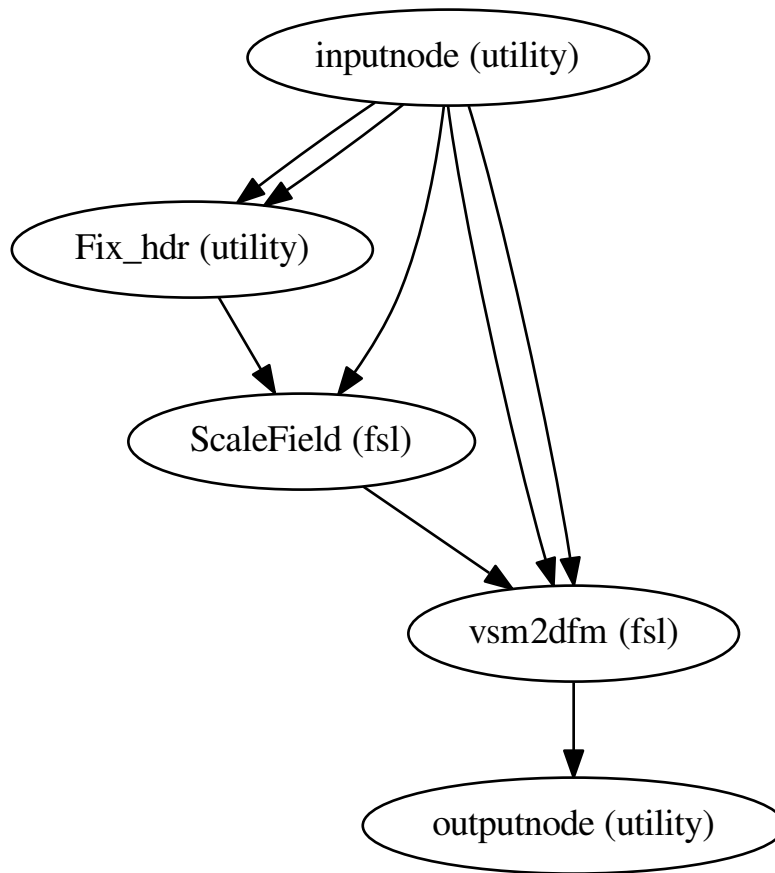
dwi_flirt()[Link to code](#)

Generates a workflow for linear registration of dwi volumes

Graph**vsm2warp()**[Link to code](#)

Converts a voxel shift map (vsm) to a displacements field (warp).

Graph



Shiftmap2Warping

add_empty_vol()[Link to code](#)

Adds an empty vol to the phase difference image

b0_average()[Link to code](#)

A function that averages the *b0* volumes from a DWI dataset. As current dMRI data are being acquired with all b-values > 0.0, the *lowb* volumes are selected by specifying the parameter *max_b*.

Warning: *b0* should be already registered (head motion artifact should be corrected).

b0_indices()

[Link to code](#)

Extract the indices of slices in a b-values file with a low b value

compute_readout()

[Link to code](#)

Computes readout time from epi params (see [eddy documentation](#)).

Warning: `params['echospacing']` should be in *sec* units.

copy_hdr()

[Link to code](#)

demean_image()

[Link to code](#)

Demean image data inside mask

eddy_rotate_bvecs()

[Link to code](#)

Rotates the input bvec file accordingly with a list of parameters sourced from eddy, as explained [here](#).

enhance()

[Link to code](#)

extract_bval()

[Link to code](#)

Writes an image containing only the volumes with b-value specified at input

hmc_split()

[Link to code](#)

Selects the reference and moving volumes from a dwi dataset for the purpose of HMC.

insert_mat()

[Link to code](#)

rads2radsec()

[Link to code](#)

Converts input phase difference map to rads

recompose_dwi()

[Link to code](#)

Recompose back the dMRI data accordingly the b-values table after EC correction

recompose_xfm()[Link to code](#)

Insert identity transformation matrices in b0 volumes to build up a list

remove_comp()[Link to code](#)Removes the volume `valid` from the 4D nifti file**reorient_bvecs()**[Link to code](#)Checks reorientations of `in_dwi` w.r.t. `old_dwi` and reorients the `in_bvec` table accordingly.**rotate_bvecs()**[Link to code](#)

Rotates the input bvec file accordingly with a list of matrices.

Note: the input affine matrix transforms points in the destination image to their corresponding coordinates in the original image. Therefore, this matrix should be inverted first, as we want to know the target position of \vec{r} .

siemens2rads()[Link to code](#)

Converts input phase difference map to rads

time_avg()[Link to code](#)

Average the input time-series, selecting the indices given in index

Warning: time steps should be already registered (corrected for head motion artifacts).

workflows.dmri.mrtrix.connectivity_mapping**create_connectivity_pipeline()**[Link to code](#)

Creates a pipeline that does the same connectivity processing as in the *dmri: Connectivity - MRtrix, CMTK, FreeSurfer* example script. Given a subject id (and completed Freesurfer reconstruction) diffusion-weighted image, b-values, and b-vectors, the workflow will return the subject's connectome as a Connectome File Format (CFF) file for use in Connectome Viewer (<http://www.cmtk.org>).

Example

```
>>> from nipy.workflows.dmri.mrtrix.connectivity_mapping import create_
    ↪connectivity_pipeline
>>> conmapper = create_connectivity_pipeline("nipy_conmap")
>>> conmapper.inputs.inputnode.subjects_dir = '.'
>>> conmapper.inputs.inputnode.subject_id = 'subj1'
```

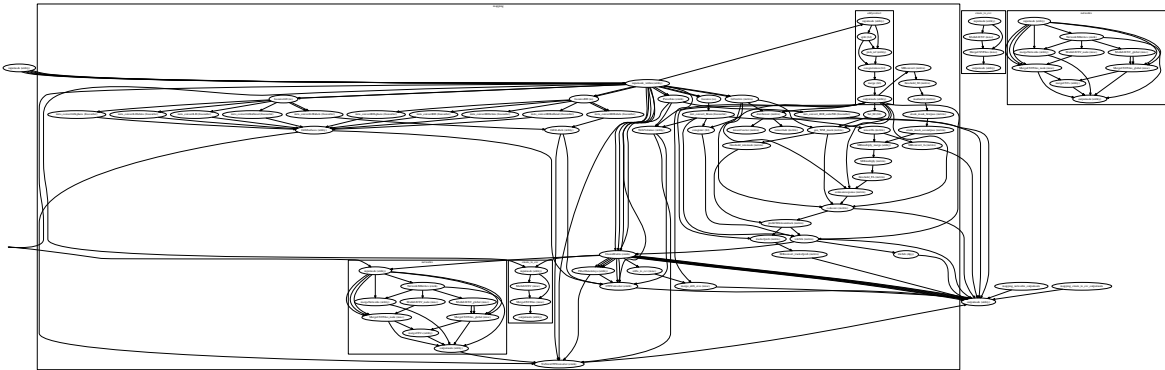
```
>>> conmapper.inputs.inputnode.dwi = 'data.nii.gz'
>>> conmapper.inputs.inputnode.bvecs = 'bvecs'
>>> conmapper.inputs.inputnode.bvals = 'bvals'
>>> conmapper.run()
```

Inputs:

```
inputnode.subject_id
inputnode.subjects_dir
inputnode.dwi
inputnode.bvecs
inputnode.bvals
inputnode.resolution_network_file
```

Outputs:

```
outputnode.connectome
outputnode.cmatrix
outputnode.networks
outputnode.fa
outputnode.struct
outputnode.tracts
outputnode.rois
outputnode.odfs
outputnode.filtered_tractography
outputnode.tdi
outputnode.nxstatscff
outputnode.nxcsv
outputnode.cmatrices_csv
outputnode.mean_fiber_length
outputnode.median_fiber_length
outputnode.fiber_length_std
```

Graph**workflows.dmri.mrtrix.diffusion****create_mrtrix_dti_pipeline()****Link to code**

Creates a pipeline that does the same diffusion processing as in the `../users/examples/dmri_mrtrix_dti` example script. Given a diffusion-weighted image, b-values, and b-vectors, the workflow will return the tractography computed from spherical deconvolution and probabilistic streamline tractography

Example

```
>>> dti = create_mrtrix_dti_pipeline("mrtrix_dti")
>>> dti.inputs.inputnode.dwi = 'data.nii'
>>> dti.inputs.inputnode.bvals = 'bvals'
>>> dti.inputs.inputnode.bvecs = 'bvecs'
>>> dti.run()
```

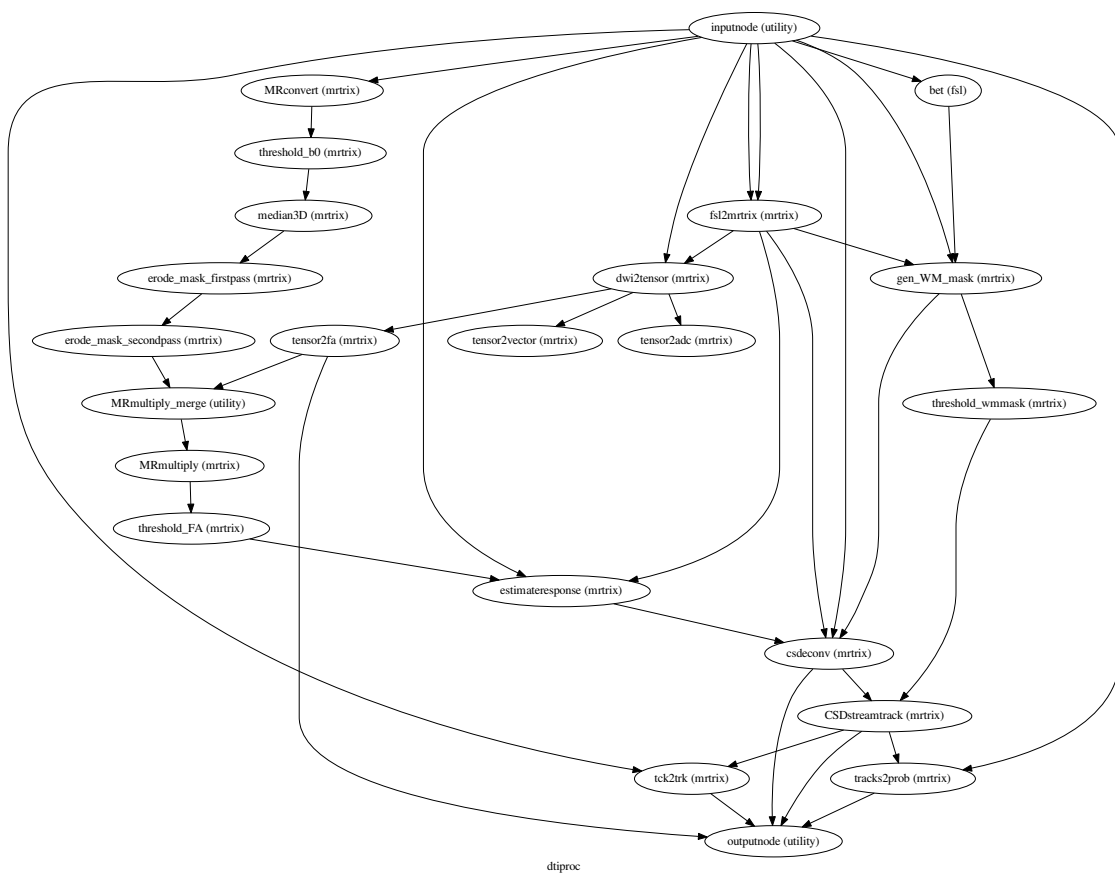
Inputs:

```
inputnode.dwi
inputnode.bvecs
inputnode.bvals
```

Outputs:

```
outputnode.fa
outputnode.tdi
outputnode.tracts_tck
outputnode.tracts_trk
outputnode.csdeconv
```

Graph



workflows.dmri.mrtrix.group_connectivity

create_group_connectivity_pipeline()

[Link to code](#)

Creates a pipeline that performs MRtrix structural connectivity processing on groups of subjects. Given a diffusion-weighted image, and text files containing the associated b-values and b-vectors, the workflow will return each subjects' connectomes in a Connectome File Format (CFF) file, for use in Connectome Viewer (<http://www.cmtk.org>).

Example

```
>>> import nipy.interfaces.freesurfer as fs
>>> import nipy.workflows.dmri.mrtrix.group_connectivity as groupwork
>>> import cmp
>>> from nipy.testing import example_data
>>> subjects_dir = '.'
>>> data_dir = '.'
>>> output_dir = '.'
>>> fs.FSCommand.set_default_subjects_dir(subjects_dir)
>>> group_list = {}
>>> group_list['group1'] = ['subj1', 'subj2']
>>> group_list['group2'] = ['subj3', 'subj4']
>>> template_args = dict(dwi=[['subject_id', 'dwi']], bvecs=[['subject_id', 'bvecs',
↳']], bvals=[['subject_id', 'bvals']])
>>> group_id = 'group1'
>>> llpipeline = groupwork.create_group_connectivity_pipeline(group_list, group_
↳id, data_dir, subjects_dir, output_dir, template_args)
>>> parcellation_name = 'scale500'
>>> llpipeline.inputs.connectivity.mapping.Parcellate.parcellation_name = _
↳parcellation_name
>>> cmp_config = cmp.configuration.PipelineConfiguration()
>>> cmp_config.parcellation_scheme = "Lausanne2008"
>>> llpipeline.inputs.connectivity.mapping.inputnode_within.resolution_network_
↳file = cmp_config.get_lausanne_parcellation('Lausanne2008')[parcellation_name][
↳'node_information_graphml']
>>> llpipeline.run()
```

Inputs:

```
group_list: Dictionary of subject lists, keyed by group name
group_id: String containing the group name
data_dir: Path to the data directory
subjects_dir: Path to the Freesurfer 'subjects' directory
output_dir: Path for the output files
template_args_dict: Dictionary of template arguments for the connectivity_
↳pipeline datasource
                    e.g.    info = dict(dwi=[['subject_id', 'dwi']],
                                       bvecs=[['subject_id', 'bvecs']],
                                       bvals=[['subject_id', 'bvals']])
```

workflows.fmri.fsl.estimate

create_fixed_effects_flow()

[Link to code](#)

Create a fixed-effects workflow

This workflow is used to combine registered copes and varcopes across runs for an individual subject

Example

```

>>> fixedfx = create_fixed_effects_flow()
>>> fixedfx.base_dir = '.'
>>> fixedfx.inputs.inputspec.copes = [['cope1run1.nii.gz', 'cope1run2.nii.gz'], [
↪ 'cope2run1.nii.gz', 'cope2run2.nii.gz']] # per contrast
>>> fixedfx.inputs.inputspec.varcopes = [['varcope1run1.nii.gz', 'varcope1run2.
↪ nii.gz'], ['varcope2run1.nii.gz', 'varcope2run2.nii.gz']] # per contrast
>>> fixedfx.inputs.inputspec.dof_files = ['dofrun1', 'dofrun2'] # per run
>>> fixedfx.run()

```

Inputs:

```

inputspec.copes : list of list of cope files (one list per contrast)
inputspec.varcopes : list of list of varcope files (one list per
                    contrast)
inputspec.dof_files : degrees of freedom files for each run

```

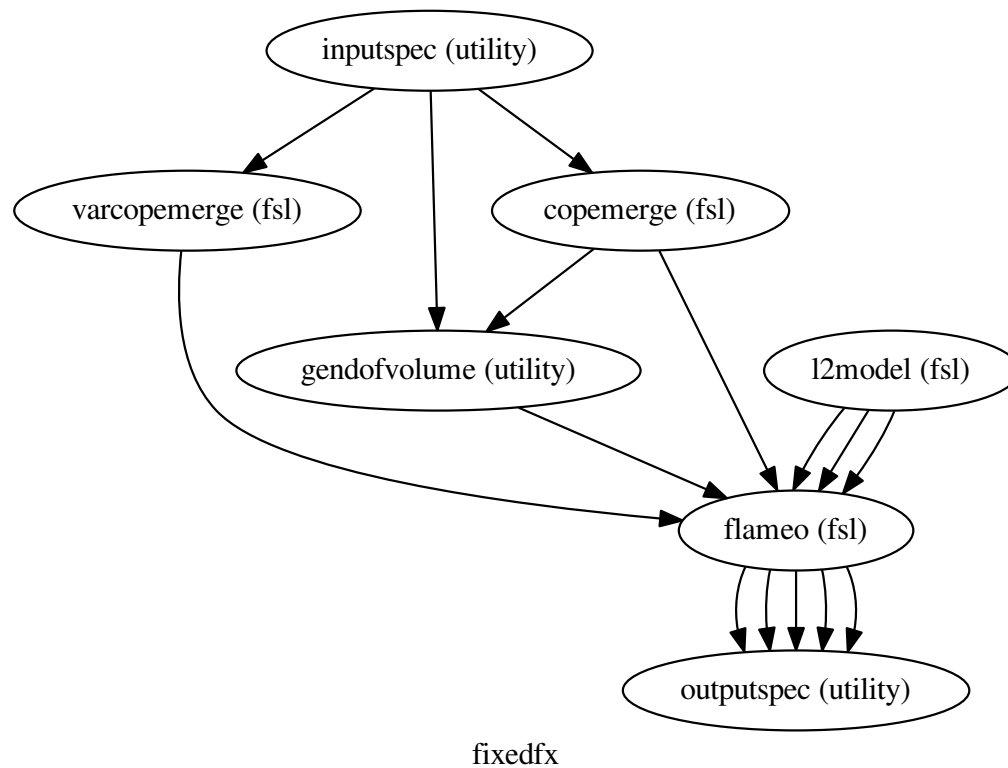
Outputs:

```

outputspec.res4d : 4d residual time series
outputspec.copes : contrast parameter estimates
outputspec.varcopes : variance of contrast parameter estimates
outputspec.zstats : z statistics of contrasts
outputspec.tstats : t statistics of contrasts

```

Graph



create_modelfit_workflow()

[Link to code](#)

Create an FSL individual modelfitting workflow

Example

```

>>> modelfit = create_modelfit_workflow()
>>> modelfit.base_dir = '.'
>>> info = dict()
>>> modelfit.inputs.inputspec.session_info = info
>>> modelfit.inputs.inputspec.interscan_interval = 3.
>>> modelfit.inputs.inputspec.film_threshold = 1000
>>> modelfit.run()

```

Inputs:

```

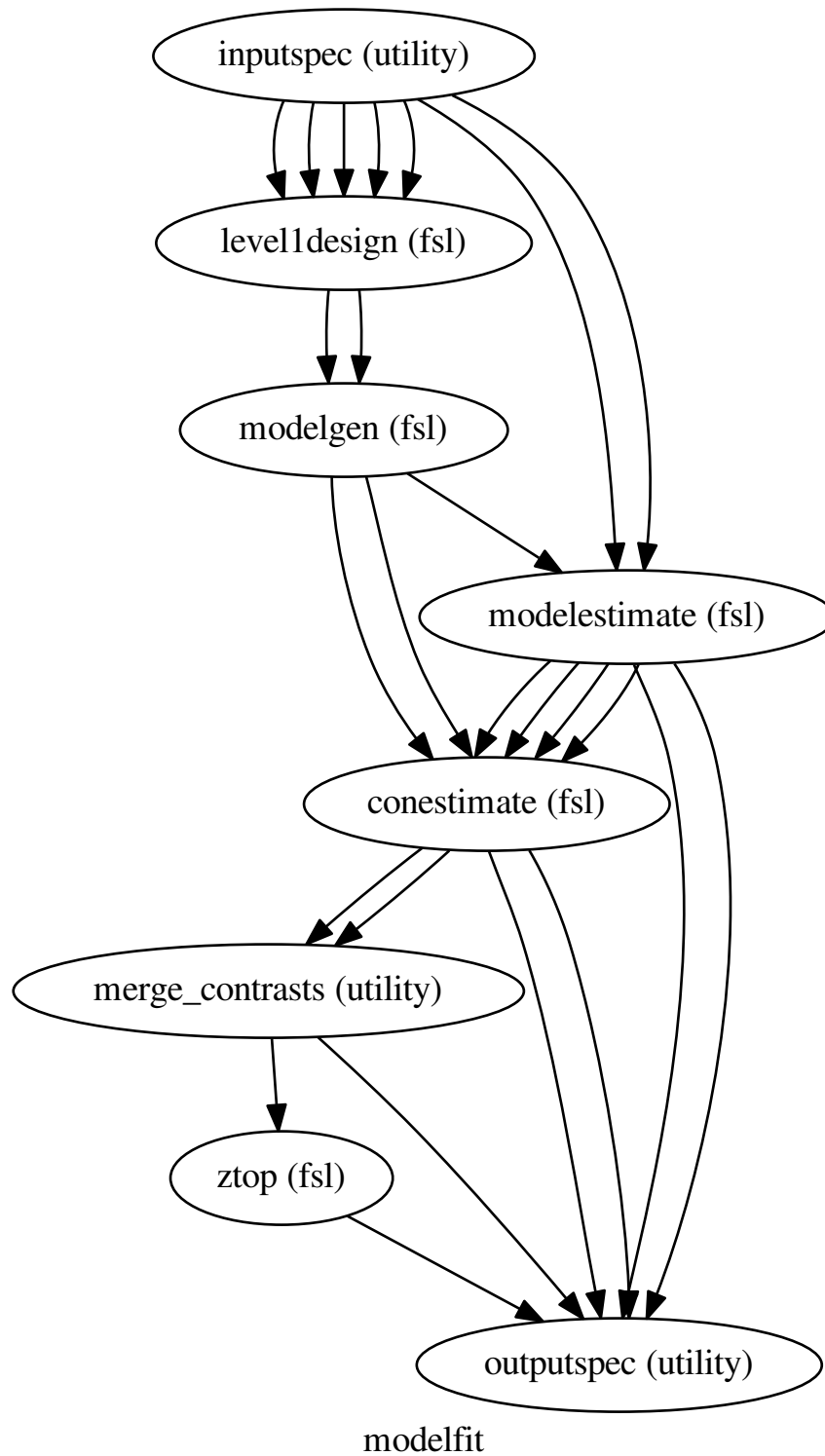
inputspec.session_info : info generated by modelgen.SpecifyModel
inputspec.interscan_interval : interscan interval
inputspec.contrasts : list of contrasts
inputspec.film_threshold : image threshold for FILM estimation
inputspec.model_serial_correlations
inputspec.bases

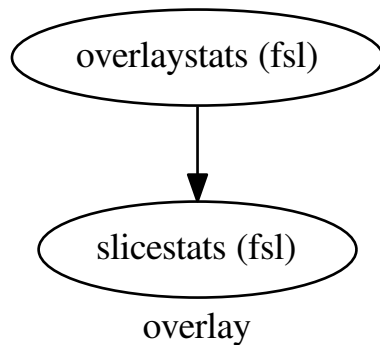
```

Outputs:

```
outputspec.copes  
outputspec.varcopes  
outputspec.dof_file  
outputspec.pfiles  
outputspec.zfiles  
outputspec.parameter_estimates
```

Graph



create_overlay_workflow()[Link to code](#)[Setup overlay workflow](#)**Graph****workflows.fmri.fsl.preprocess****create_featreg_preproc()**[Link to code](#)

Create a FEAT preprocessing workflow with registration to one volume of the first run

Parameters

```

name : name of workflow (default: featpreproc)
highpass : boolean (default: True)
whichvol : which volume of the first run to register to ('first', 'middle', 'last
↳', 'mean')

```

Inputs:

```

inputspec.func : functional runs (filename or list of filenames)
inputspec.fwhm : fwhm for smoothing with SUSAN
inputspec.highpass : HWHM in TRs (if created with highpass=True)

```

Outputs:

```

outputspec.reference : volume to which runs are realigned
outputspec.motion_parameters : motion correction parameters
outputspec.realigned_files : motion corrected files
outputspec.motion_plots : plots of motion correction parameters
outputspec.mask : mask file used to mask the brain
outputspec.smoothed_files : smoothed functional data
outputspec.highpassed_files : highpassed functional data (if highpass=True)
outputspec.mean : mean file

```

Example

```
>>> preproc = create_featreg_preproc()
>>> preproc.inputs.inputsfunc = ['f3.nii', 'f5.nii']
>>> preproc.inputs.inputsfunc.fwhm = 5
>>> preproc.inputs.inputsfunc.highpass = 128./(2*2.5)
>>> preproc.base_dir = '/tmp'
>>> preproc.run()
```

```
>>> preproc = create_featreg_preproc(highpass=False, whichvol='mean')
>>> preproc.inputs.inputsfunc = 'f3.nii'
>>> preproc.inputs.inputsfunc.fwhm = 5
>>> preproc.base_dir = '/tmp'
>>> preproc.run()
```


7.2. workflows.fmri.fsl.preprocess



`create_fsl_fs_preproc()`

[Link to code](#)

Create a FEAT preprocessing workflow together with freesurfer

Parameters

```
name : name of workflow (default: preproc)
highpass : boolean (default: True)
whichvol : which volume of the first run to register to ('first', 'middle', 'mean
→')
```

Inputs:

```
inputspec.func : functional runs (filename or list of filenames)
inputspec.fwhm : fwhm for smoothing with SUSAN
inputspec.highpass : HWHM in TRs (if created with highpass=True)
inputspec.subject_id : freesurfer subject id
inputspec.subjects_dir : freesurfer subjects dir
```

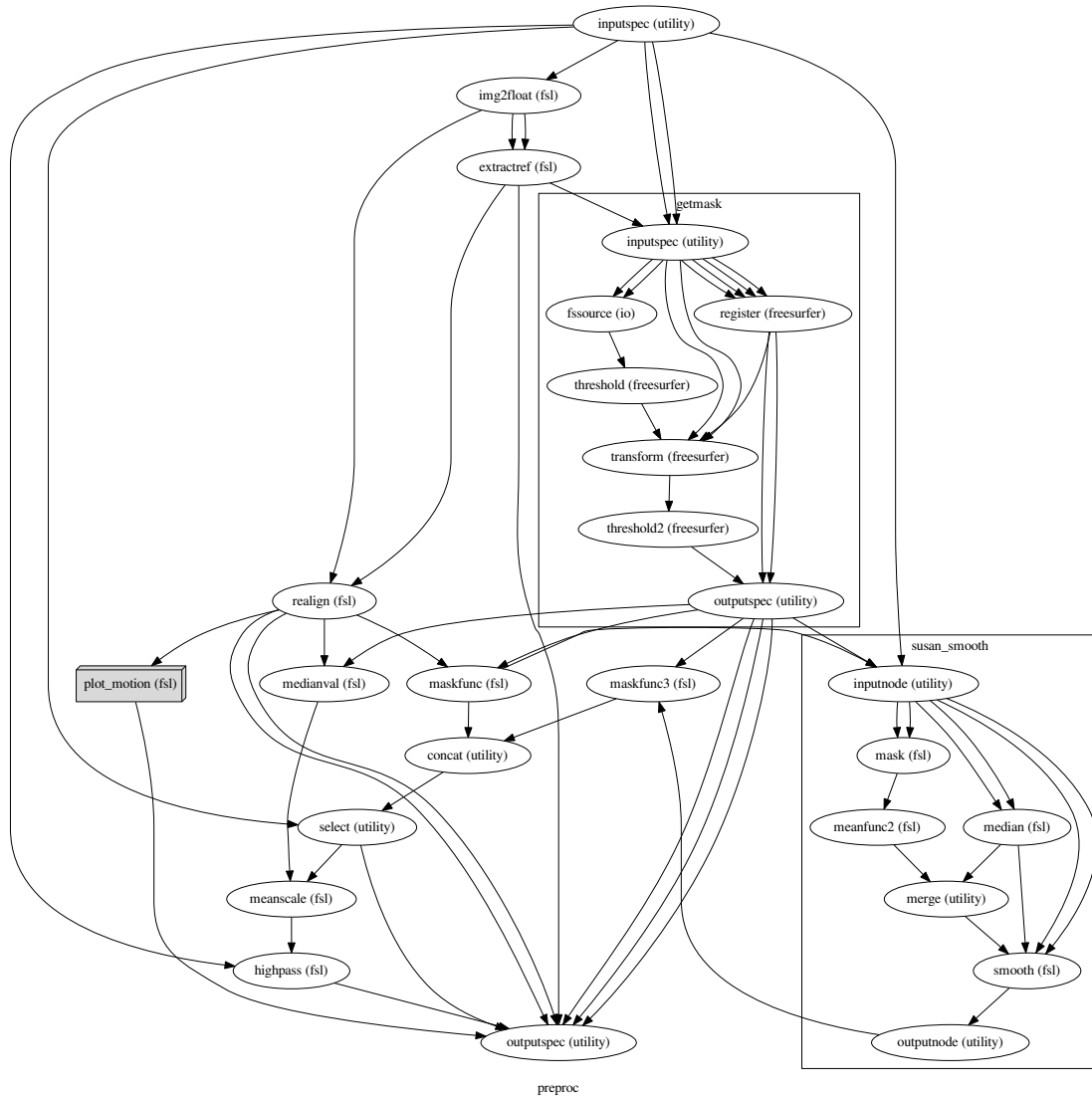
Outputs:

```
outputspec.reference : volume to which runs are realigned
outputspec.motion_parameters : motion correction parameters
outputspec.realigned_files : motion corrected files
outputspec.motion_plots : plots of motion correction parameters
outputspec.mask_file : mask file used to mask the brain
outputspec.smoothed_files : smoothed functional data
outputspec.highpassed_files : highpassed functional data (if highpass=True)
outputspec.reg_file : bregister registration files
outputspec.reg_cost : bregister registration cost files
```

Example

```
>>> preproc = create_fsl_fs_preproc(whichvol='first')
>>> preproc.inputs.inputspec.highpass = 128./(2*2.5)
>>> preproc.inputs.inputspec.func = ['f3.nii', 'f5.nii']
>>> preproc.inputs.inputspec.subjects_dir = '.'
>>> preproc.inputs.inputspec.subject_id = 's1'
>>> preproc.inputs.inputspec.fwhm = 6
>>> preproc.run()
```

Graph

`create_parallelfat_preproc()`[Link to code](#)

Preprocess each run with FSL independently of the others

Parameters

name : name of workflow (default: featpreproc)
 highpass : boolean (default: **True**)

Inputs:

inputspec.func : functional runs (filename **or** list of filenames)
 inputspec.fwhm : fwhm **for** smoothing **with** SUSAN
 inputspec.highpass : HWHM **in** TRs (**if** created **with** highpass=**True**)

Outputs:

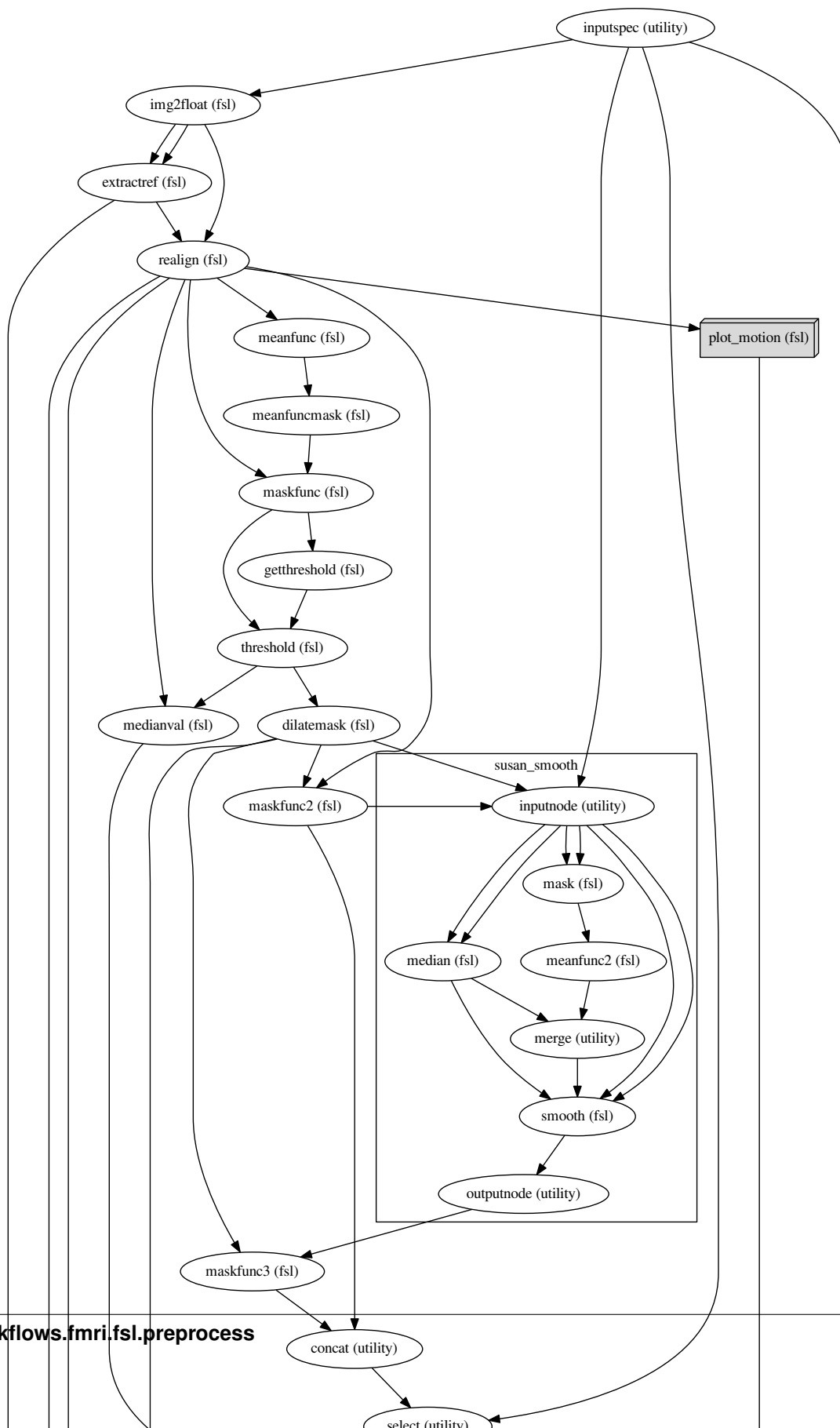
```
outputspec.reference : volume to which runs are realigned
outputspec.motion_parameters : motion correction parameters
outputspec.realigned_files : motion corrected files
outputspec.motion_plots : plots of motion correction parameters
outputspec.mask : mask file used to mask the brain
outputspec.smoothed_files : smoothed functional data
outputspec.highpassed_files : highpassed functional data (if highpass=True)
outputspec.mean : mean file
```

Example

```
>>> preproc = create_parallelfcat_preproc()
>>> preproc.inputs.inputsfunc = ['f3.nii', 'f5.nii']
>>> preproc.inputs.inputsfunc.fwhm = 5
>>> preproc.inputs.inputsfunc.highpass = 128./(2*2.5)
>>> preproc.base_dir = '/tmp'
>>> preproc.run()
```

```
>>> preproc = create_parallelfcat_preproc(highpass=False)
>>> preproc.inputs.inputsfunc = 'f3.nii'
>>> preproc.inputs.inputsfunc.fwhm = 5
>>> preproc.base_dir = '/tmp'
>>> preproc.run()
```

Graph



`create_susan_smooth()`

[Link to code](#)

Create a SUSAN smoothing workflow

Parameters

```
name : name of workflow (default: susan_smooth)
separate_masks : separate masks for each run
```

Inputs:

```
inputnode.in_files : functional runs (filename or list of filenames)
inputnode.fwhm : fwhm for smoothing with SUSAN
inputnode.mask_file : mask used for estimating SUSAN thresholds (but not for
↳smoothing)
```

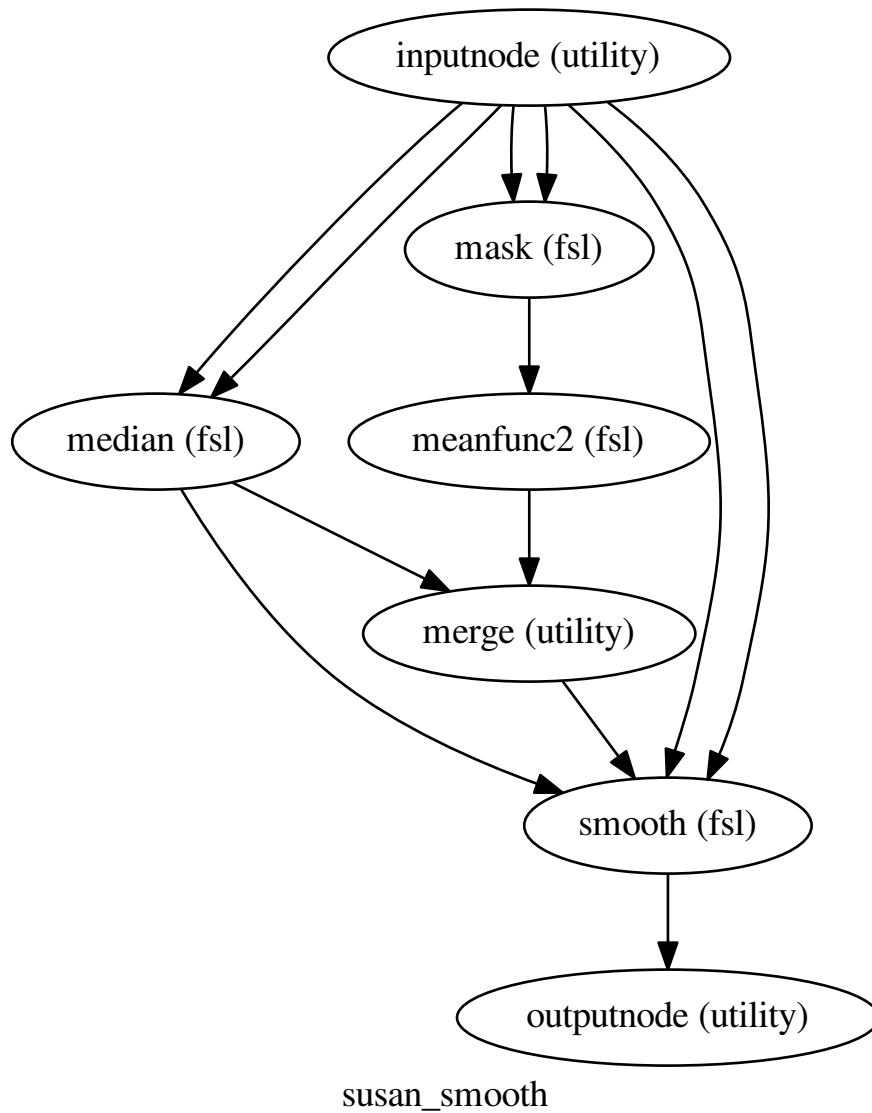
Outputs:

```
outputnode.smoothed_files : functional runs (filename or list of filenames)
```

Example

```
>>> smooth = create_susan_smooth()
>>> smooth.inputs.inputnode.in_files = 'f3.nii'
>>> smooth.inputs.inputnode.fwhm = 5
>>> smooth.inputs.inputnode.mask_file = 'mask.nii'
>>> smooth.run()
```

Graph



chooseindex()

[Link to code](#)

create_reg_workflow()

[Link to code](#)

Create a FEAT preprocessing workflow

Parameters

```
name : name of workflow (default: 'registration')
```

Inputs:

```
inputspec.source_files : files (filename or list of filenames to register)
inputspec.mean_image : reference image to use
inputspec.anatomical_image : anatomical image to coregister to
inputspec.target_image : registration target
```

Outputs:

```
outputspec.func2anat_transform : FLIRT transform
outputspec.anat2target_transform : FLIRT+FNIRT transform
outputspec.transformed_files : transformed files in target space
outputspec.transformed_mean : mean image in target space
```

Example

`getbtthresh()`

[Link to code](#)

`getmeanscale()`

[Link to code](#)

`getthresshop()`

[Link to code](#)

`getusans()`

[Link to code](#)

`pickfirst()`

[Link to code](#)

`pickmiddle()`

[Link to code](#)

`pickvol()`

[Link to code](#)

`workflows.fmri.spm.preprocess`

`create_DARTEL_template()`

[Link to code](#)

Create a vbm workflow that generates DARTEL-based template

Example

```
>>> preproc = create_DARTEL_template()
>>> preproc.inputs.inputs.spec.structural_files = [os.path.abspath('s1.nii'), os.
↳ path.abspath('s3.nii')]
>>> preproc.inputs.inputs.spec.template_prefix = 'Template'
>>> preproc.run()
```

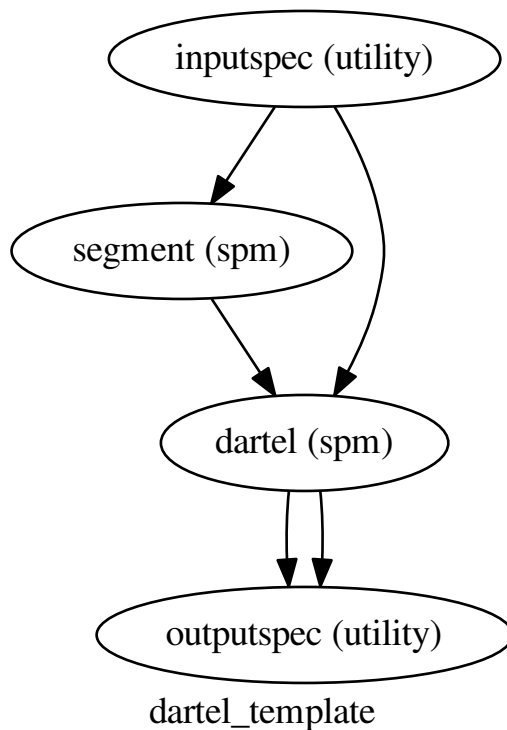
Inputs:

```
inputs.spec.structural_files : structural data to be used to create templates
inputs.spec.template_prefix : prefix for dartel template
```

Outputs:

```
outputs.spec.template_file : DARTEL template
outputs.spec.flow_fields : warps from input struct files to the template
```

Graph



`create_spm_preproc()`

[Link to code](#)

Create an spm preprocessing workflow with freesurfer registration and artifact detection.

The workflow realigns and smooths and registers the functional images with the subject's freesurfer space.

Example

```
>>> preproc = create_spm_preproc()
>>> preproc.base_dir = '.'
>>> preproc.inputs.inputspec.fwhm = 6
>>> preproc.inputs.inputspec.subject_id = 's1'
>>> preproc.inputs.inputspec.subjects_dir = '.'
>>> preproc.inputs.inputspec.functionals = ['f3.nii', 'f5.nii']
>>> preproc.inputs.inputspec.norm_threshold = 1
>>> preproc.inputs.inputspec.zintensity_threshold = 3
```

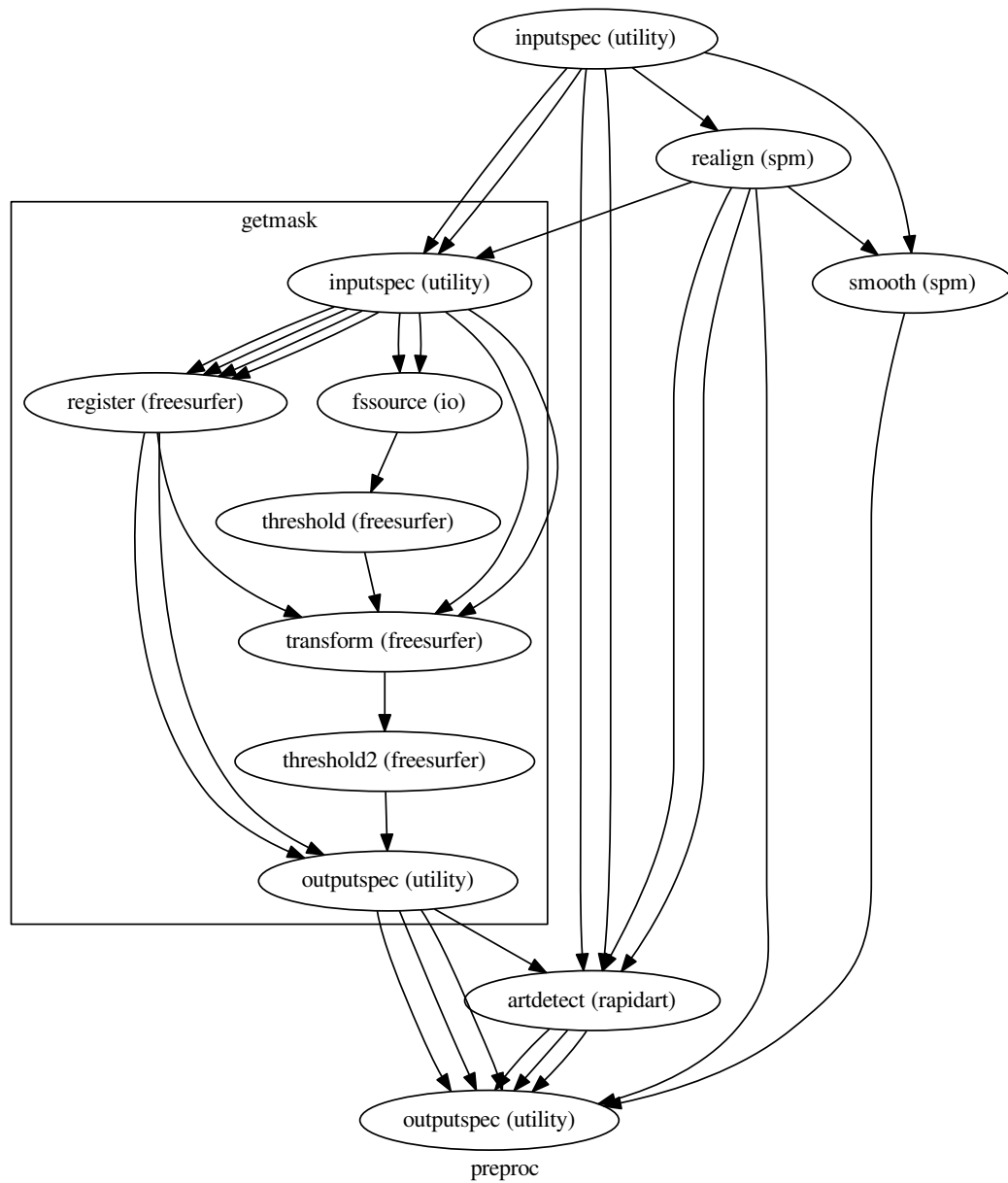
Inputs:

```
inputspec.functionals : functional runs use 4d nifti
inputspec.subject_id : freesurfer subject id
inputspec.subjects_dir : freesurfer subjects dir
inputspec.fwhm : smoothing fwhm
inputspec.norm_threshold : norm threshold for outliers
inputspec.zintensity_threshold : intensity threshold in z-score
```

Outputs:

```
outputspec.realignment_parameters : realignment parameter files
outputspec.smoothed_files : smoothed functional files
outputspec.outlier_files : list of outliers
outputspec.outlier_stats : statistics of outliers
outputspec.outlier_plots : images of outliers
outputspec.mask_file : binary mask file in reference image space
outputspec.reg_file : registration file that maps reference image to
                      freesurfer space
outputspec.reg_cost : cost of registration (useful for detecting misalignment)
```

Graph

`create_vbm_preproc()`

[Link to code](#)

Create a vbm workflow that generates DARTEL-based warps to MNI space

Based on: <http://www.fil.ion.ucl.ac.uk/~john/misc/VBMclass10.pdf>

Example

```
>>> preproc = create_vbm_preproc()
>>> preproc.inputs.inputspec.fwhm = 8
>>> preproc.inputs.inputspec.structural_files = [os.path.abspath('s1.nii'), os.
↳ path.abspath('s3.nii')]
>>> preproc.inputs.inputspec.template_prefix = 'Template'
>>> preproc.run()
```

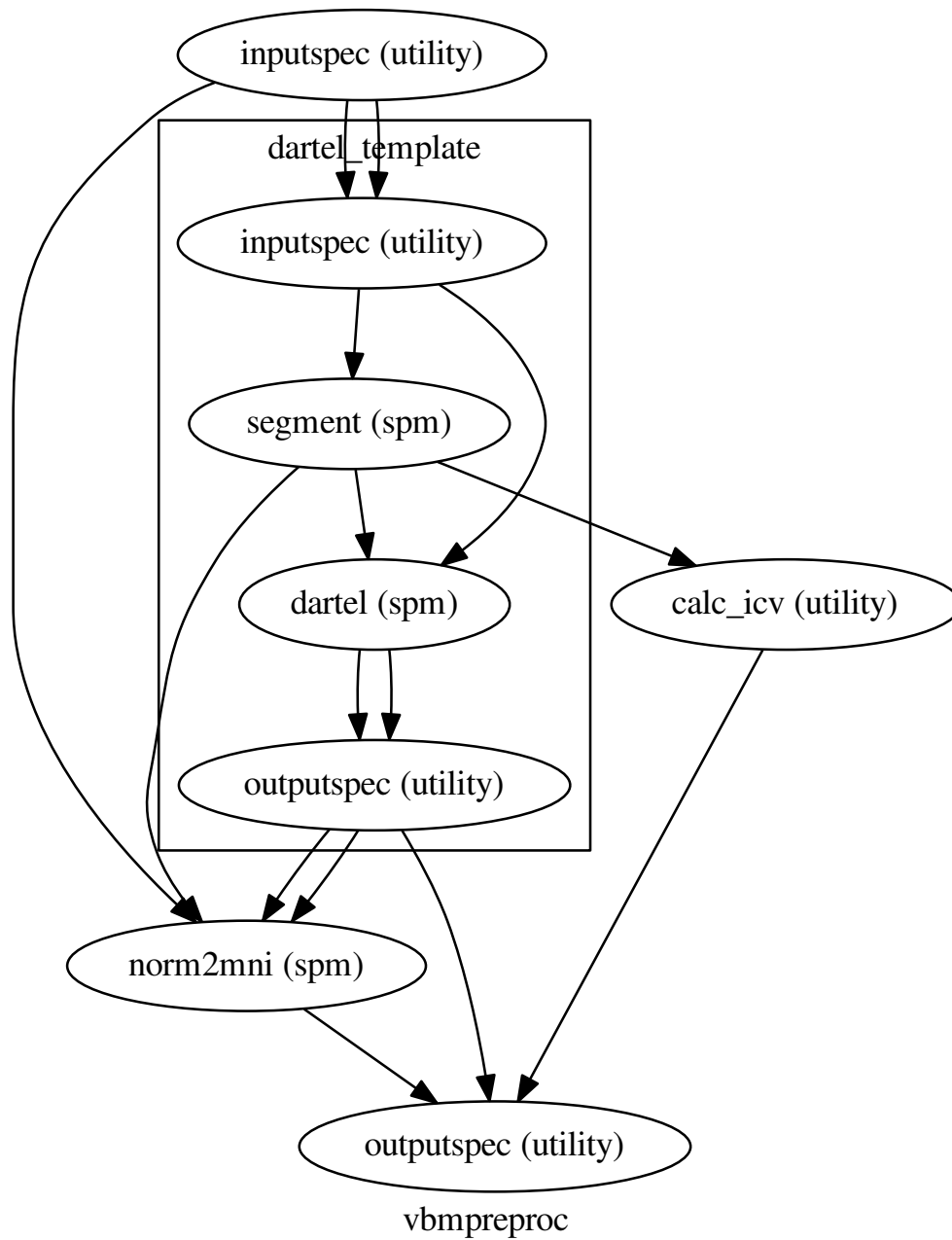
Inputs:

```
inputspec.structural_files : structural data to be used to create templates
inputspec.fwhm: single of triplet for smoothing when normalizing to MNI space
inputspec.template_prefix : prefix for dartel template
```

Outputs:

```
outputspec.normalized_files : normalized gray matter files
outputspec.template_file : DARTEL template
outputspec.icv : intracranial volume (cc - assuming dimensions in mm)
```

Graph



workflows.misc.utils

`get_affine()`

[Link to code](#)

`get_data_dims()`

[Link to code](#)

`get_vox_dims()`

[Link to code](#)

`id_list_from_lookup_table()`

[Link to code](#)

`region_list_from_volume()`

[Link to code](#)

`select_aparc()`

[Link to code](#)

`select_aparc_annot()`

[Link to code](#)

workflows.rsfmri.fsl.resting

`create_realign_flow()`

[Link to code](#)

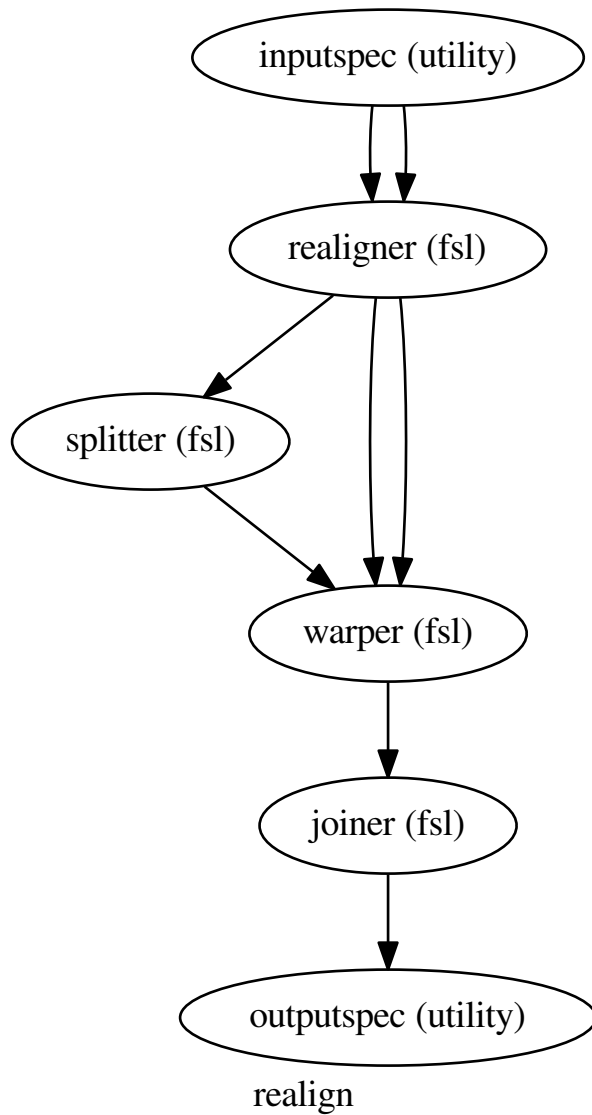
Realign a time series to the middle volume using spline interpolation

Uses MCFLIRT to realign the time series and ApplyWarp to apply the rigid body transformations using spline interpolation (unknown order).

Example

```
>>> wf = create_realign_flow()
>>> wf.inputs.inputs.spec.func = 'f3.nii'
>>> wf.run()
```

Graph



`create_resting_preproc()`

[Link to code](#)

Create a “resting” time series preprocessing workflow
The noise removal is based on Behzadi et al. (2007)

Parameters

name : name of workflow (default: restpreproc)

Inputs:

```
inputspec.func : functional run (filename or list of filenames)
```

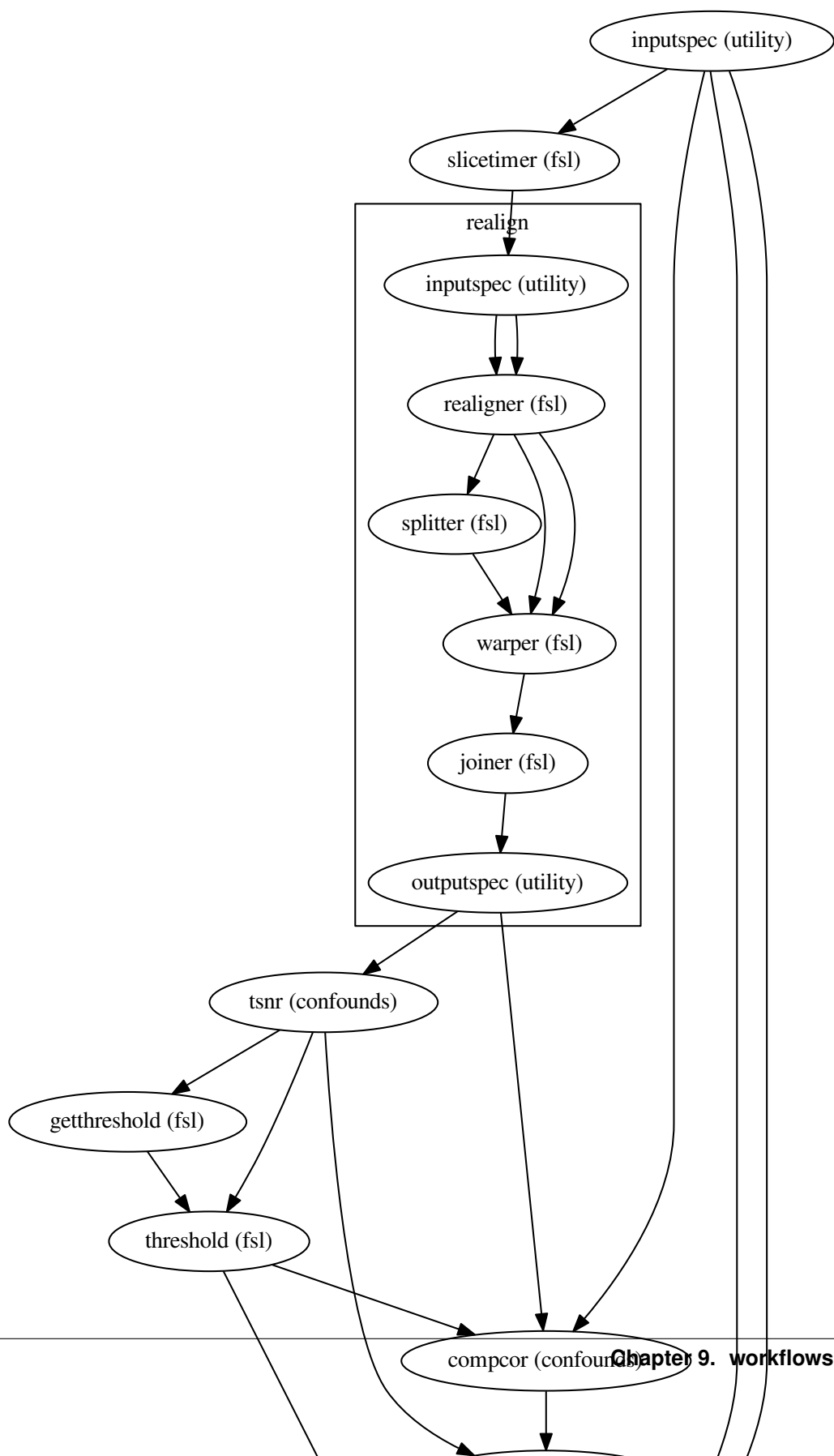
Outputs:

```
outputspec.noise_mask_file : voxels used for PCA to derive noise components  
outputspec.filtered_file : bandpass filtered and noise-reduced time series
```

Example

```
>>> TR = 3.0  
>>> wf = create_resting_preproc()  
>>> wf.inputs.inputspec.func = 'f3.nii'  
>>> wf.inputs.inputspec.num_noise_components = 6  
>>> wf.inputs.inputspec.highpass_sigma = 100/(2*TR)  
>>> wf.inputs.inputspec.lowpass_sigma = 12.5/(2*TR)  
>>> wf.run()
```

Graph



`select_volume()`

[Link to code](#)

Return the middle index of a file

workflows.smri.ants.ANTSBuildTemplate

ANTSTemplateBuildSingleIterationWF()

[Link to code](#)

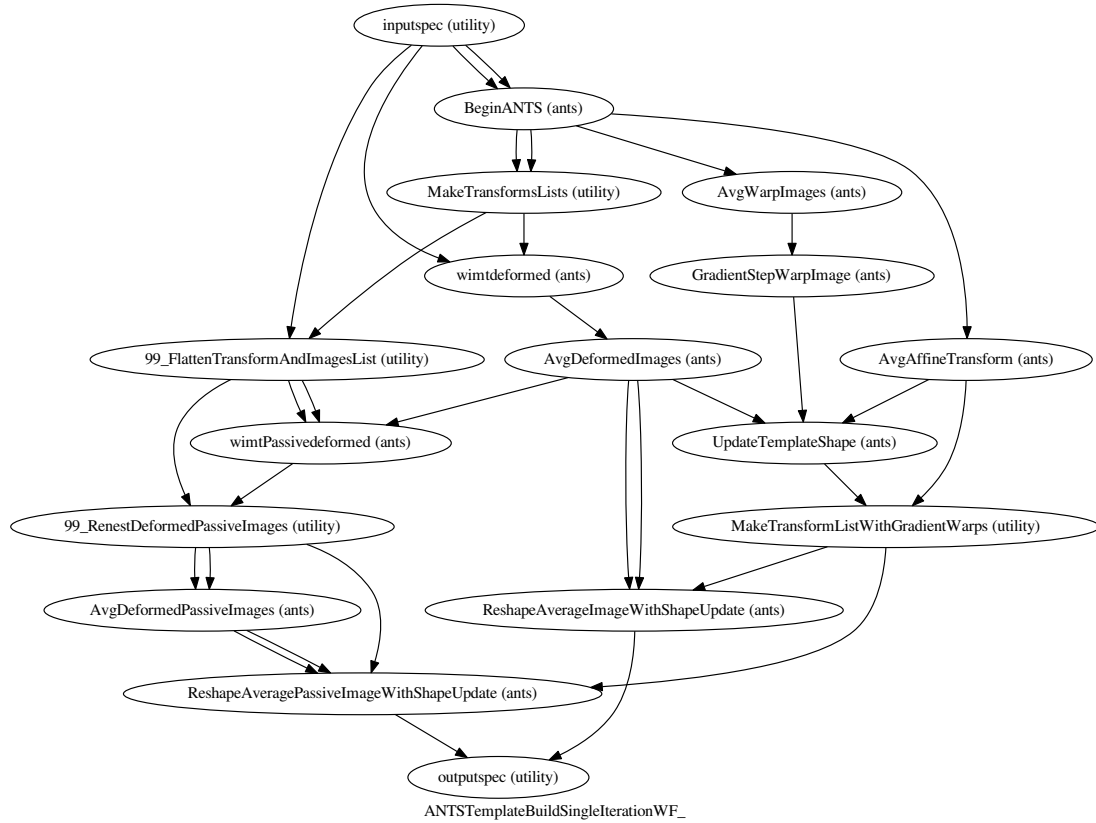
Inputs:

```
inputspec.images :  
inputspec.fixed_image :  
inputspec.ListOfPassiveImagesDictionaries :
```

Outputs:

```
outputspec.template :  
outputspec.transforms_list :  
outputspec.passive_deformed_templates :
```

Graph



FlattenTransformAndImagesList()

[Link to code](#)

GetFirstListElement()

[Link to code](#)

MakeListsOfTransformLists()

[Link to code](#)

MakeTransformListWithGradientWarps()

[Link to code](#)

RenestDeformedPassiveImages()

[Link to code](#)

workflows.smri.ants.antsRegistrationBuildTemplate

antsRegistrationTemplateBuildSingleIterationWF ()

[Link to code](#)

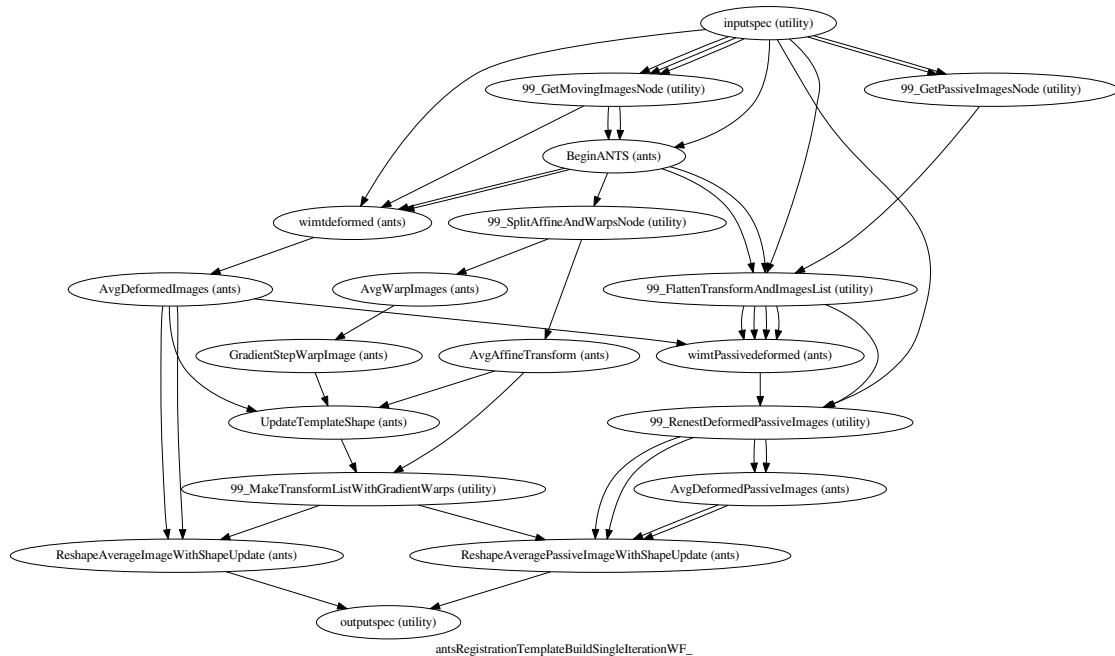
Inputs:

```
inputspec.images :
inputspec.fixed_image :
inputspec.ListOfPassiveImagesDictionaries :
inputspec.interpolationMapping :
```

Outputs:

```
outputspec.template :
outputspec.transforms_list :
outputspec.passive_deformed_templates :
```

Graph



FlattenTransformAndImagesList ()

[Link to code](#)

GetFirstListElement ()

[Link to code](#)

GetMovingImages ()

[Link to code](#)

This currently ONLY works when registrationImageTypes has length of exactly 1. When the new multi-variate registration is introduced, it will be expanded.

`GetPassiveImages ()`

[Link to code](#)

`MakeTransformListWithGradientWarps ()`

[Link to code](#)

`RenestDeformedPassiveImages ()`

[Link to code](#)

`SplitAffineAndWarpComponents ()`

[Link to code](#)

`makeListOfOneElement ()`

[Link to code](#)

`workflows.smri.freesurfer.autorecon1`

`checkT1s ()`

[Link to code](#)

Verifying size of inputs and setting workflow parameters

`create_AutoRecon1 ()`

[Link to code](#)

Creates the AutoRecon1 workflow in nipy.

Inputs:: inputspec.T1_files : T1 files (mandatory) inputspec.T2_file : T2 file (optional) inputspec.FLAIR_file : FLAIR file (optional) inputspec.cw256 : Conform inputs to 256 FOV (optional) inputspec.num_threads: Number of threads to use with EM Register (default=1)

Output:

`workflows.smri.freesurfer.autorecon2`

`copy_ltas ()`

[Link to code](#)

`workflows.smri.freesurfer.bem`

`create_bem_flow ()`

[Link to code](#)

Uses MNE's Watershed algorithm to create Boundary Element Meshes (BEM) for a subject's brain, inner/outer skull, and skin. The surfaces are returned in the desired (by default, stereolithic .stl) format.

Example

```
>>> from nipy.workflows.smri.freesurfer import create_bem_flow
>>> bemflow = create_bem_flow()
>>> bemflow.inputs.inputs.spec.subject_id = 'subj1'
>>> bemflow.inputs.inputs.spec.subjects_dir = '.'
>>> bemflow.run()
```

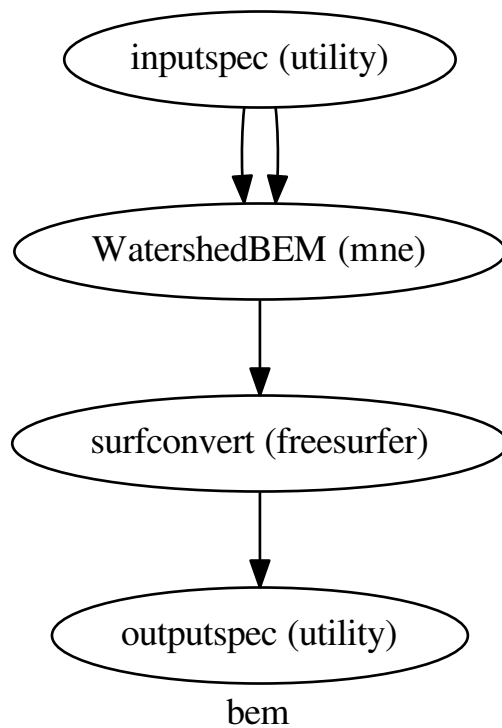
Inputs:

```
inputs.spec.subject_id : freesurfer subject id
inputs.spec.subjects_dir : freesurfer subjects directory
```

Outputs:

```
outputs.meshes : output boundary element meshes in (by default)
↳ stereolithographic (.stl) format
```

Graph



workflows.smri.freesurfer.recon

create_reconall_workflow()

[Link to code](#)

Creates the ReconAll workflow in Nipype. This workflow is designed to run the same commands as FreeSurfer's reconall script but with the added features that a Nipype workflow provides. Before running this workflow, it is necessary to have the FREESURFER_HOME environmental variable set to the directory containing the version of FreeSurfer to be used in this workflow.

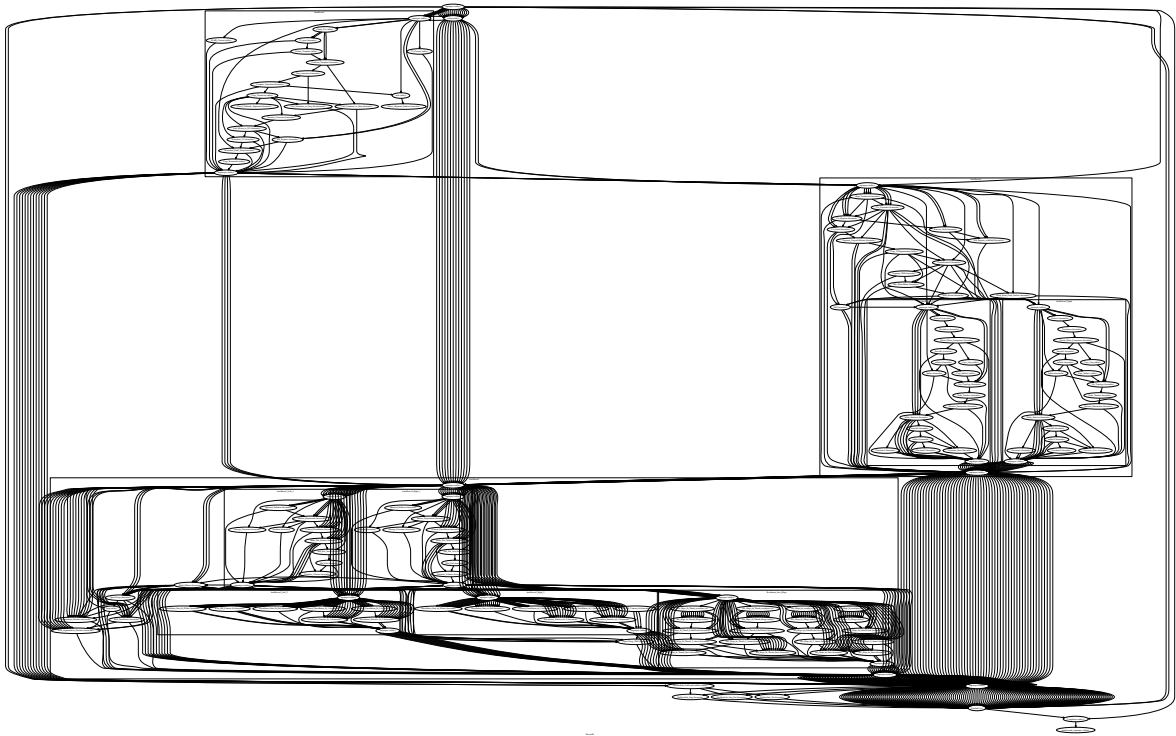
Example

```
>>> from nipype.workflows.smri.freesurfer import create_reconall_workflow
>>> recon_all = create_reconall_workflow()
>>> recon_all.inputs.inputs.spec.subject_id = 'subj1'
>>> recon_all.inputs.inputs.spec.subjects_dir = '.'
>>> recon_all.inputs.inputs.spec.T1_files = 'T1.nii.gz'
>>> recon_flow.run()
```

Inputs:: inputs.spec.subjects_dir : subjects directory (mandatory) inputs.spec.subject_id : name of subject (mandatory) inputs.spec.T1_files : T1 files (mandatory) inputs.spec.T2_file : T2 file (optional) inputs.spec.FLAIR_file : FLAIR file (optional) inputs.spec.cw256 : Conform inputs to 256 FOV (optional) inputs.spec.num_threads: Number of threads on nodes that utilize OpenMP (default=1) plugin_args : Dictionary of plugin args to set to nodes that utilize OpenMP (optional)

Outputs:: postdatasink_outputs.spec.subject_id : name of the datasinked output folder in the subjects directory
Note: The input subject_id is not passed to the commands in the workflow. Commands that require subject_id are reading implicit inputs from {SUBJECTS_DIR}/{subject_id}. For those commands the subject_id is set to the default value and SUBJECTS_DIR is set to the node directory. The implicit inputs are then copied to the node directory in order to mimic a SUBJECTS_DIR structure. For example, if the command implicitly reads in brainmask.mgz, the interface would copy that input file to {node_dir}/{subject_id}/mri/brainmask.mgz and set SUBJECTS_DIR to node_dir. The workflow only uses the input subject_id to datasink the outputs to {subjects_dir}/{subject_id}.

Graph



`create_skullstripped_recon_flow()`

[Link to code](#)

Performs recon-all on volumes that are already skull stripped. FreeSurfer fails to perform skullstripping on some volumes (especially MP2RAGE). This can be avoided by doing skullstripping before running recon-all (using for example SPECTRE algorithm).

Example

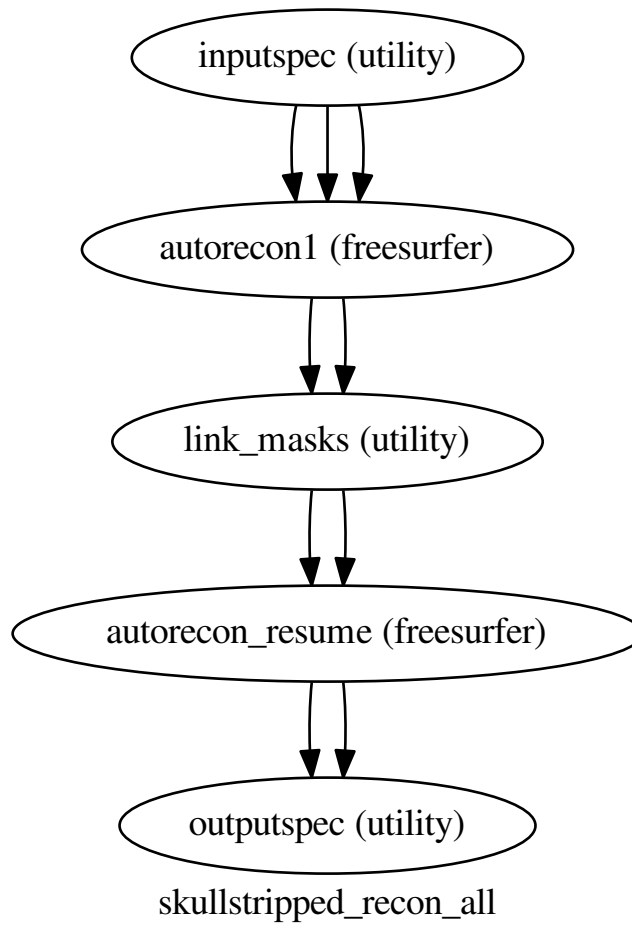
```
>>> from nipy.workflows.smri.freesurfer import create_skullstripped_recon_flow
>>> recon_flow = create_skullstripped_recon_flow()
>>> recon_flow.inputs.inputs.spec.subject_id = 'subj1'
>>> recon_flow.inputs.inputs.spec.T1_files = 'T1.nii.gz'
>>> recon_flow.run()
```

Inputs:: `inputs.spec.T1_files` : skullstripped T1_files (mandatory) `inputs.spec.subject_id` : freesurfer subject id (optional) `inputs.spec.subjects_dir` : freesurfer subjects directory (optional)

Outputs:

```
outputs.spec.subject_id : freesurfer subject id
outputs.spec.subjects_dir : freesurfer subjects directory
```

Graph



workflows.smri.freesurfer.utils

`create_get_stats_flow()`

[Link to code](#)

Retrieves stats from labels

Parameters

name [string] name of workflow

withreg [boolean] indicates whether to register source to label

Example

Inputs:

```

inputspec.source_file : reference image for mask generation
inputspec.label_file : label file from which to get ROIs

(optionally with registration)
inputspec.reg_file : bbreg file (assumes reg from source to label
inputspec.inverse : boolean whether to invert the registration
inputspec.subjects_dir : freesurfer subjects directory

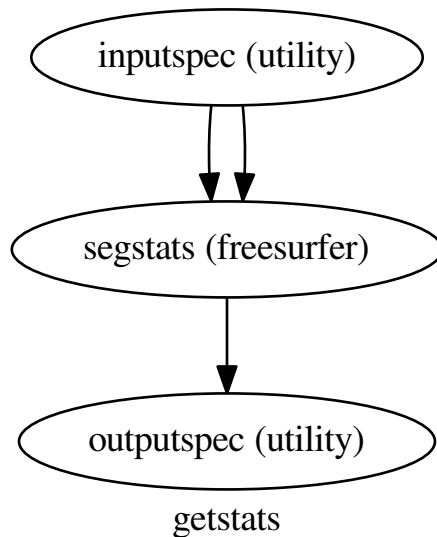
```

Outputs:

```

outputspec.stats_file : stats file

```

Graph**create_getmask_flow()**[Link to code](#)

Registers a source file to freesurfer space and create a brain mask in source space
 Requires fsl tools for initializing registration

Parameters

name [string] name of workflow
dilate_mask [boolean] indicates whether to dilate mask or not

Example

```

>>> getmask = create_getmask_flow()
>>> getmask.inputs.inputspec.source_file = 'mean.nii'

```

```
>>> getmask.inputs.inputs.spec.subject_id = 's1'
>>> getmask.inputs.inputs.spec.subjects_dir = '.'
>>> getmask.inputs.inputs.spec.contrast_type = 't2'
```

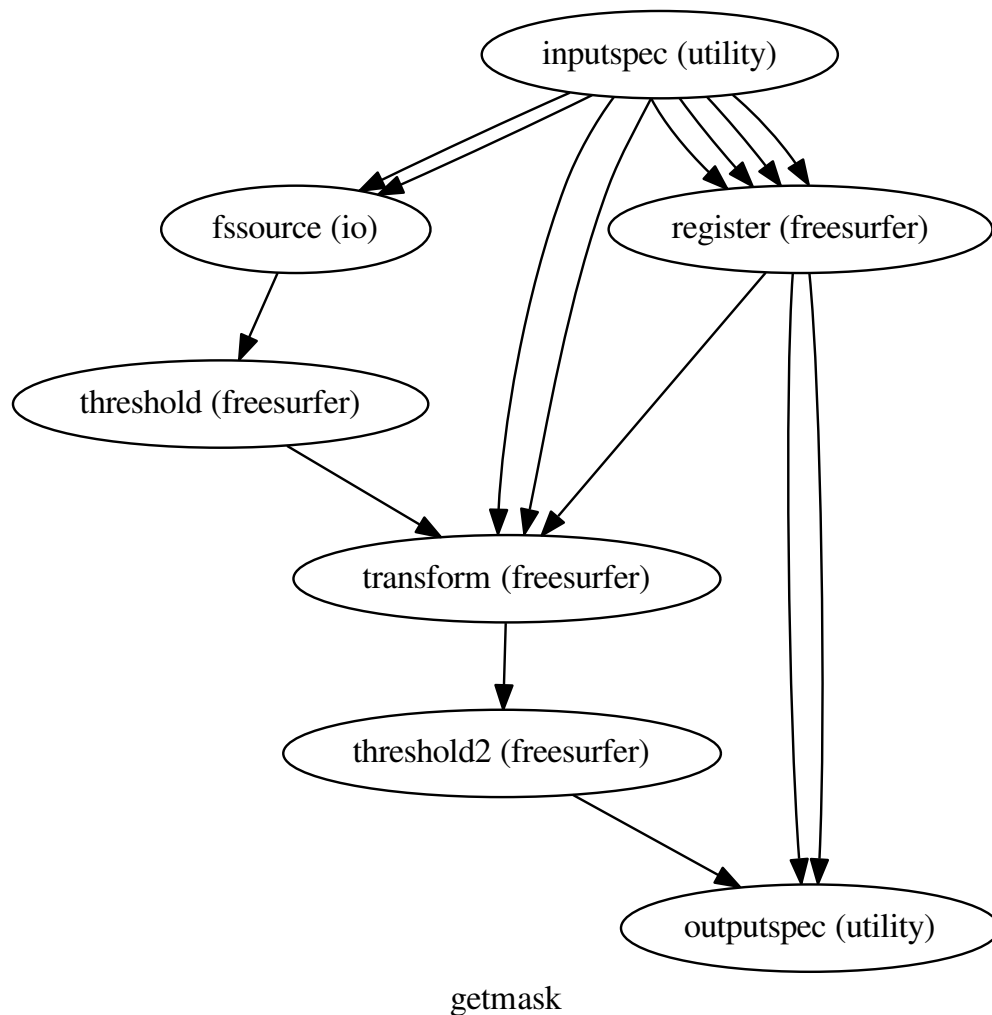
Inputs:

```
inputs.spec.source_file : reference image for mask generation
inputs.spec.subject_id : freesurfer subject id
inputs.spec.subjects_dir : freesurfer subjects directory
inputs.spec.contrast_type : MR contrast of reference image
```

Outputs:

```
outputs.spec.mask_file : binary mask file in reference image space
outputs.spec.reg_file : registration file that maps reference image to
                        freesurfer space
outputs.spec.reg_cost : cost of registration (useful for detecting misalignment)
```


Graph



create_tessellation_flow()

[Link to code](#)

Tessellates the input subject's aseg.mgz volume and returns the surfaces for each region in stereolithic (.stl) format

Example

```

>>> from nipy.workflows.smri.freesurfer import create_tessellation_flow
>>> tessflow = create_tessellation_flow()
>>> tessflow.inputs.inputspec.subject_id = 'subj1'
>>> tessflow.inputs.inputspec.subjects_dir = '.'
>>> tessflow.inputs.inputspec.lookup_file = 'FreeSurferColorLUT.txt'

```

```
>>> tessflow.run()
```

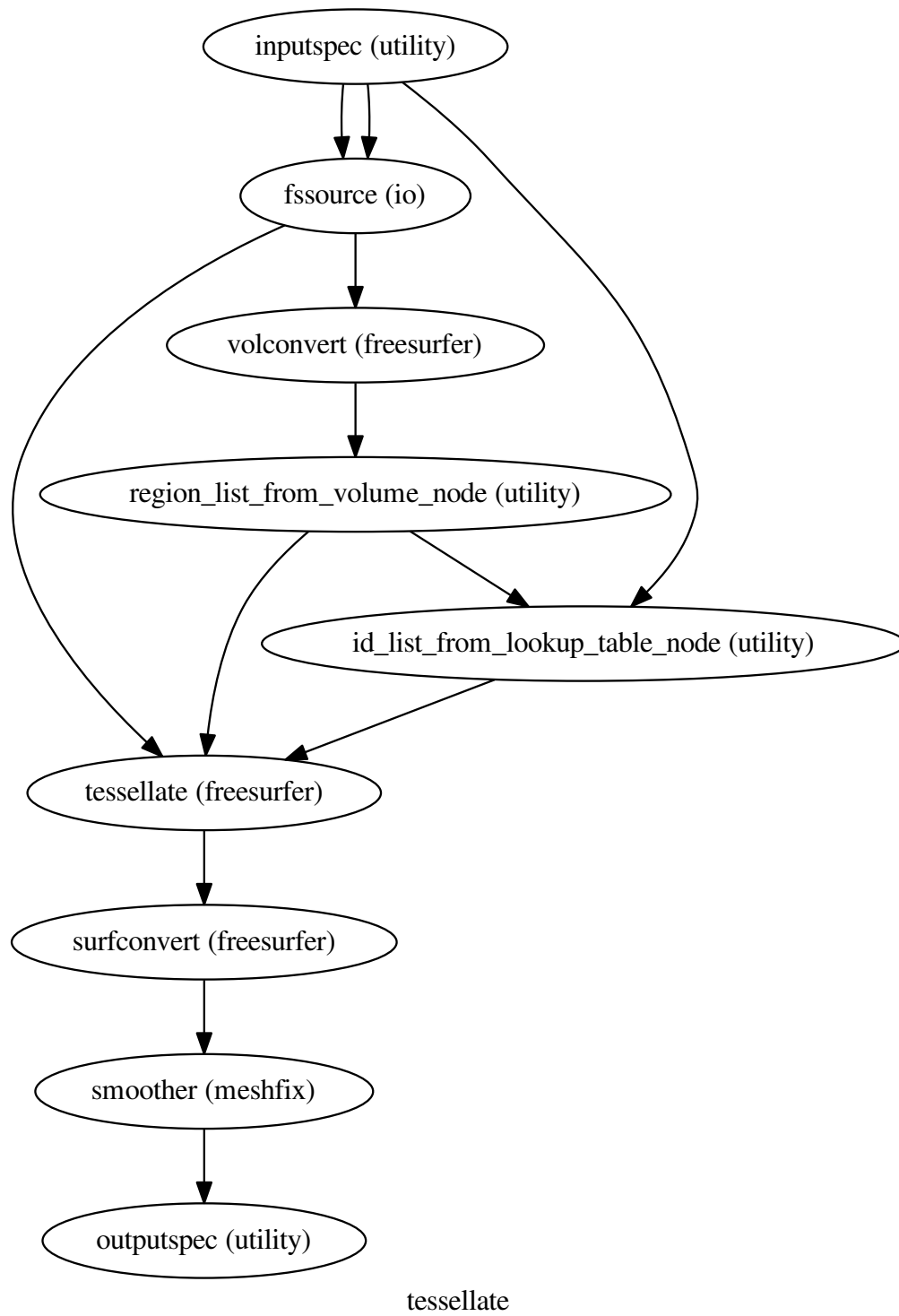
Inputs:

```
inputspec.subject_id : freesurfer subject id
inputspec.subjects_dir : freesurfer subjects directory
inputspec.lookup_file : lookup file from freesurfer directory
```

Outputs:

```
outputspec.meshes : output region meshes in (by default) stereolithographic (.
↳stl) format
```

Graph



copy_file()[Link to code](#)

Create a function to copy a file that can be modified by a following node without changing the original file.

copy_files()[Link to code](#)

Create a function to copy a file that can be modified by a following node without changing the original file

get_aparc_aseg()[Link to code](#)

Return the aparc+aseg.mgz file

getdefaultconfig()[Link to code](#)**makedirs_p()**[Link to code](#)**workflows.smri.niftyreg.groupwise****create_groupwise_average()**[Link to code](#)

Create the overall workflow that embeds all the rigid, affine and non-linear components.

Inputs:

```
inputspec.in_files - The input files to be registered
inputspec.ref_file - The initial reference image that the input files
                    are registered to
inputspec.rmask_file - Mask of the reference image
inputspec.in_trans_files - Initial transformation files (affine or
                    cpps)
```

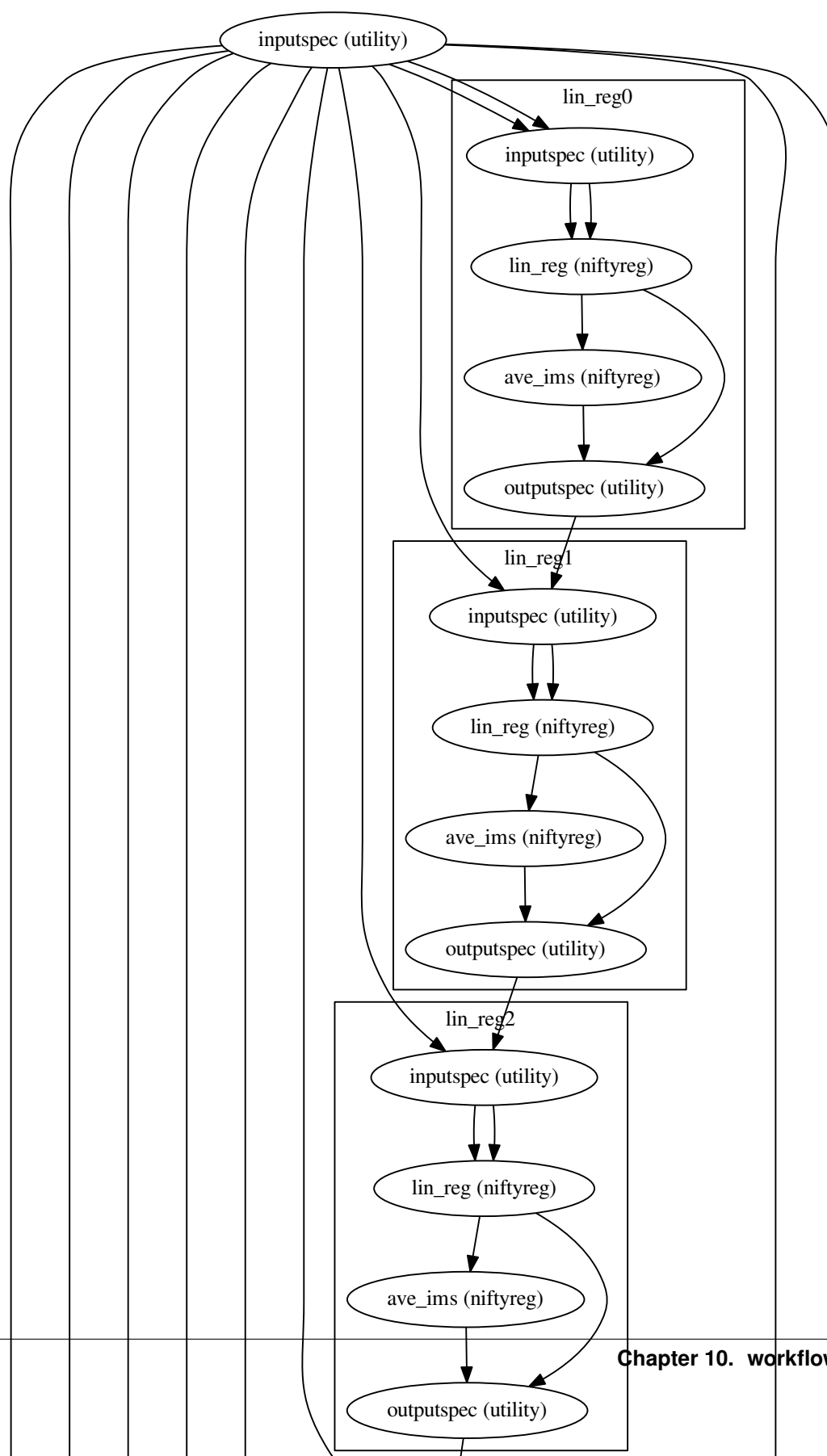
Outputs:

```
outputspec.average_image - The average image
outputspec.cpp_files - The bspline transformation files
```

Example

```
>>> from nipy.workflows.smri.niftyreg import create_groupwise_average
>>> node = create_groupwise_average('groupwise_av')
>>> node.inputs.inputspec.in_files = [
...     'file1.nii.gz', 'file2.nii.gz']
>>> node.inputs.inputspec.ref_file = ['ref.nii.gz']
>>> node.inputs.inputspec.rmask_file = ['mask.nii.gz']
>>> node.run()
```


Graph



create_linear_gw_step()

[Link to code](#)

Creates a workflow that performs linear co-registration of a set of images using RegAladin, producing an average image and a set of affine transformation matrices linking each of the floating images to the average.

Inputs:

```

inputspec.in_files - The input files to be registered
inputspec.ref_file - The initial reference image that the input files
                    are registered to
inputspec.rmask_file - Mask of the reference image
inputspec.in_aff_files - Initial affine transformation files

```

Outputs:

```

outputspec.average_image - The average image
outputspec.aff_files - The affine transformation files

```

Optional arguments:

```

linear_options_hash - An options dictionary containing a list of
                    parameters for RegAladin that take
                    the same form as given in the interface (default None)
demean - Selects whether to demean the transformation matrices when
         performing the averaging (default True)
initial_affines - Selects whether to iterate over initial affine
                 images, which we generally won't have (default False)

```

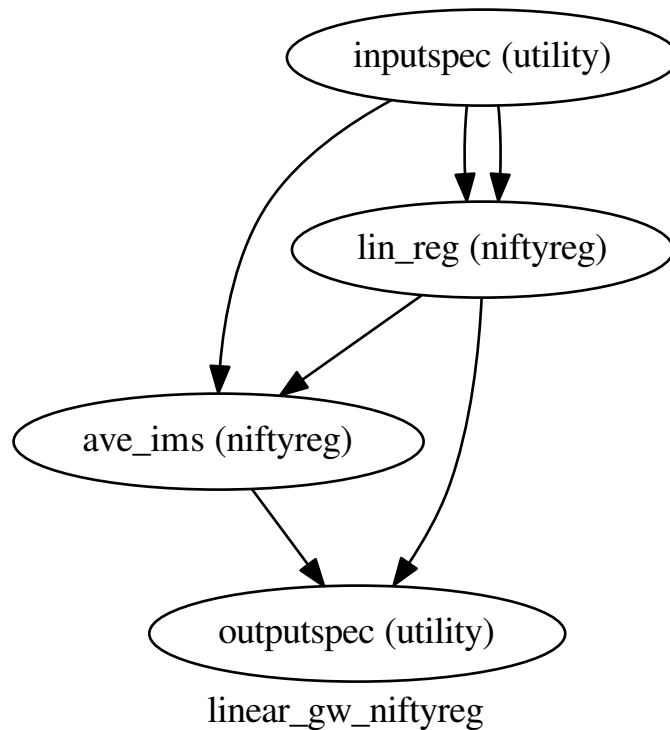
Example

```

>>> from nipy.workflows.smri.niftyreg import create_linear_gw_step
>>> lgw = create_linear_gw_step('my_linear_coreg')
>>> lgw.inputs.inputspec.in_files = [
...     'file1.nii.gz', 'file2.nii.gz']
>>> lgw.inputs.inputspec.ref_file = ['ref.nii.gz']
>>> lgw.run()

```

Graph



create_nonlinear_gw_step()

[Link to code](#)

Creates a workflow that perform non-linear co-registrations of a set of images using RegF3d, producing an non-linear average image and a set of cpp transformation linking each of the floating images to the average.

Inputs:

```

inputspec.in_files - The input files to be registered
inputspec.ref_file - The initial reference image that the input files
                    are registered to
inputspec.rmask_file - Mask of the reference image
inputspec.in_trans_files - Initial transformation files (affine or
                        cpps)

```

Outputs:

```

outputspec.average_image - The average image
outputspec.cpp_files - The bspline transformation files

```

Optional arguments:

```

nonlinear_options_hash - An options dictionary containing a list of
                        parameters for RegAladin that take the
                        same form as given in the interface (default None)

```

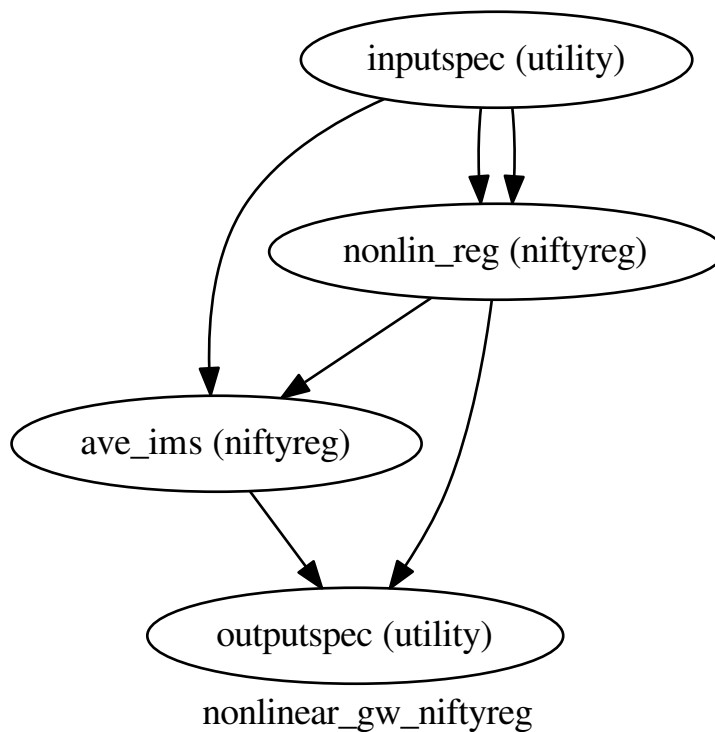


```
initial_affines - Selects whether to iterate over initial affine
                  images, which we generally won't have (default False)
```

Example

```
>>> from nipy.workflows.smri.niftyreg import create_nonlinear_gw_step
>>> nlc = create_nonlinear_gw_step('nonlinear_coreg')
>>> nlc.inputs.inputspec.in_files = [
...     'file1.nii.gz', 'file2.nii.gz']
>>> nlc.inputs.inputspec.ref_file = ['ref.nii.gz']
>>> nlc.run()
```

Graph



- Examples

Introduction

This script, `camino_dti_tutorial.py`, demonstrates the ability to perform basic diffusion analysis in a Nipype pipeline:

```
python dmri_camino_dti.py
```

We perform this analysis using the FSL course data, which can be acquired from here: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

Import necessary modules from nipype.

```
import os                                # system functions
import nipype.interfaces.io as nio       # Data i/o
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe      # pypeline engine
import nipype.interfaces.camino as camino
import nipype.interfaces.fsl as fsl
import nipype.interfaces.camino2trackvis as cam2trk
import nipype.algorithms.misc as misc
```

We use the following functions to scrape the voxel and data dimensions of the input images. This allows the pipeline to be flexible enough to accept and process images of varying size. The SPM Face tutorial (`fmri_spm_face.py`) also implements this inferral of voxel size from the data.

```
def get_vox_dims(volume):
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    voxdims = hdr.get_zooms()
    return [float(voxdims[0]), float(voxdims[1]), float(voxdims[2])]

def get_data_dims(volume):
    import nibabel as nb
    from nipype.utils import NUMPY_MMAP
```

```

if isinstance(volume, list):
    volume = volume[0]
nii = nb.load(volume, mmap=NUMPY_MMAP)
hdr = nii.header
datadims = hdr.get_data_shape()
return [int(datadims[0]), int(datadims[1]), int(datadims[2])]

def get_affine(volume):
    import nibabel as nb
    from nipy.utils import NUMPY_MMAP
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    return nii.affine

subject_list = ['subj1']
fsl.FSLCommand.set_default_output_type('NIFTI')

```

Map field names to individual subject runs

```

info = dict(dwi=[['subject_id', 'data']],
            bvecs=[['subject_id', 'bvecs']],
            bvals=[['subject_id', 'bvals']])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                    name="infosource")

```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipy.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipy.pipeline.engine.Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```

datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                              outfields=list(info.keys())),
                    name='datasource')

datasource.inputs.template = "%s/%s"

# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
datasource.inputs.base_directory = os.path.abspath('fsl_course_data/fdt/')

datasource.inputs.field_template = dict(dwi='%s/%s.nii.gz')
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

```

An `inputnode` is used to pass the data obtained by the data grabber to the actual processing functions

```

inputnode = pe.Node(interface=util.IdentityInterface(fields=["dwi", "bvecs",
↪ "bvals"]), name="inputnode")

```

Setup for Diffusion Tensor Computation

In this section we create the nodes necessary for diffusion analysis. First, the diffusion image is converted to voxel order.

```
image2voxel = pe.Node(interface=camino.Image2Voxel(), name="image2voxel")
fsl2scheme = pe.Node(interface=camino.FSL2Scheme(), name="fsl2scheme")
fsl2scheme.inputs.usegradmod = True
```

Second, diffusion tensors are fit to the voxel-order data.

```
dtifit = pe.Node(interface=camino.DTIFit(), name='dtifit')
```

Next, a lookup table is generated from the schemefile and the signal-to-noise ratio (SNR) of the unweighted (q=0) data.

```
dtlutgen = pe.Node(interface=camino.DTLUTGen(), name="dtlutgen")
dtlutgen.inputs.snr = 16.0
dtlutgen.inputs.inversion = 1
```

In this tutorial we implement probabilistic tractography using the PICO algorithm. PICO tractography requires an estimate of the fibre direction and a model of its uncertainty in each voxel; this is produced using the following node.

```
picopdfs = pe.Node(interface=camino.PicoPDFs(), name="picopdfs")
picopdfs.inputs.inputmodel = 'dt'
```

An FSL BET node creates a brain mask is generated from the diffusion image for seeding the PICO tractography.

```
bet = pe.Node(interface=fsl.BET(), name="bet")
bet.inputs.mask = True
```

Finally, tractography is performed. First DT streamline tractography.

```
trackdt = pe.Node(interface=camino.TrackDT(), name="trackdt")
```

Now camino's Probabilistic Index of connectivity algorithm. In this tutorial, we will use only 1 iteration for time-saving purposes.

```
trackpico = pe.Node(interface=camino.TrackPICO(), name="trackpico")
trackpico.inputs.iterations = 1
```

Currently, the best program for visualizing tracts is TrackVis. For this reason, a node is included to convert the raw tract data to .trk format. Solely for testing purposes, another node is added to perform the reverse.

```
cam2trk_dt = pe.Node(interface=cam2trk.Camino2Trackvis(), name="cam2trk_dt")
cam2trk_dt.inputs.min_length = 30
cam2trk_dt.inputs.voxel_order = 'LAS'

cam2trk_pico = pe.Node(interface=cam2trk.Camino2Trackvis(), name="cam2trk_pico")
cam2trk_pico.inputs.min_length = 30
cam2trk_pico.inputs.voxel_order = 'LAS'

trk2camino = pe.Node(interface=cam2trk.Trackvis2Camino(), name="trk2camino")
```

Tracts can also be converted to VTK and OOGL formats, for use in programs such as GeomView and Paraview, using the following two nodes. For VTK use VtkStreamlines.

```
procstreamlines = pe.Node(interface=camino.ProcStreamlines(), name=
↳ "procstreamlines")
procstreamlines.inputs.outputtracts = 'oogl'
```

We can also produce a variety of scalar values from our fitted tensors. The following nodes generate the fractional anisotropy and diffusivity trace maps and their associated headers.

```
fa = pe.Node(interface=camino.ComputeFractionalAnisotropy(), name='fa')
trace = pe.Node(interface=camino.ComputeTensorTrace(), name='trace')
dteig = pe.Node(interface=camino.ComputeEigensystem(), name='dteig')

analyzeheader_fa = pe.Node(interface=camino.AnalyzeHeader(), name="analyzeheader_
↪fa")
analyzeheader_fa.inputs.datatype = "double"
analyzeheader_trace = analyzeheader_fa.clone('analyzeheader_trace')

fa2nii = pe.Node(interface=misc.CreateNifti(), name='fa2nii')
trace2nii = fa2nii.clone("trace2nii")
```

Since we have now created all our nodes, we can now define our workflow and start making connections.

```
tractography = pe.Workflow(name='tractography')

tractography.connect([(inputnode, bet, [("dwi", "in_file")])])
```

File format conversion

```
tractography.connect([(inputnode, image2voxel, [("dwi", "in_file")]),
                      (inputnode, fsl2scheme, [("bvecs", "bvec_file"),
                                                ("bvals", "bval_file")])
                      ])
```

Tensor fitting

```
tractography.connect([(image2voxel, dtifit, [['voxel_order', 'in_file']]),
                      (fsl2scheme, dtifit, [['scheme', 'scheme_file']])
                      ])
```

Workflow for applying DT streamline tractogpahy

```
tractography.connect([(bet, trackdt, [("mask_file", "seed_file")])])
tractography.connect([(dtifit, trackdt, [("tensor_fitted", "in_file")])])
```

Workflow for applying PICO

```
tractography.connect([(bet, trackpico, [("mask_file", "seed_file")])])
tractography.connect([(fsl2scheme, dtlutgen, [("scheme", "scheme_file")])])
tractography.connect([(dtlutgen, picopdfs, [("dtLUT", "luts")])])
tractography.connect([(dtifit, picopdfs, [("tensor_fitted", "in_file")])])
tractography.connect([(picopdfs, trackpico, [("pdfs", "in_file")])])

# ProcStreamlines might throw memory errors - comment this line out in such case
tractography.connect([(trackdt, procstreamlines, [("tracked", "in_file")])])
```

Connecting the Fractional Anisotropy and Trace nodes is simple, as they obtain their input from the tensor fitting.

This is also where our voxel- and data-grabbing functions come in. We pass these functions, along with the original DWI image from the input node, to the header-generating nodes. This ensures that the files will be correct and readable.

```
tractography.connect([(dtifit, fa, [("tensor_fitted", "in_file")])])
tractography.connect([(fa, analyzeheader_fa, [("fa", "in_file")])])
tractography.connect([(inputnode, analyzeheader_fa, [(('dwi', get_vox_dims),
↪'voxel_dims'),
                                                    (('dwi', get_data_dims),
↪'data_dims')])])])
```

```

tractography.connect([(fa, fa2nii, [('fa', 'data_file')])])
tractography.connect([(inputnode, fa2nii, [('dwi', get_affine), 'affine'])])
tractography.connect([(analyzeheader_fa, fa2nii, [('header', 'header_file')])])

tractography.connect([(dtifit, trace, [("tensor_fitted", "in_file")])])
tractography.connect([(trace, analyzeheader_trace, [("trace", "in_file")])])
tractography.connect([(inputnode, analyzeheader_trace, [(('dwi', get_vox_dims),
↪ 'voxel_dims'),
                                                                    (('dwi', get_data_dims),
↪ 'data_dims')])])
tractography.connect([(trace, trace2nii, [('trace', 'data_file')])])
tractography.connect([(inputnode, trace2nii, [(('dwi', get_affine), 'affine')])])
tractography.connect([(analyzeheader_trace, trace2nii, [('header', 'header_file
↪ ')])])

tractography.connect([(dtifit, dteig, [("tensor_fitted", "in_file")])])

tractography.connect([(trackpico, cam2trk_pico, [('tracked', 'in_file')])])
tractography.connect([(trackdt, cam2trk_dt, [('tracked', 'in_file')])])
tractography.connect([(inputnode, cam2trk_pico, [(('dwi', get_vox_dims), 'voxel_
↪ dims'),
                                                                    (('dwi', get_data_dims), 'data_
↪ dims')])])

tractography.connect([(inputnode, cam2trk_dt, [(('dwi', get_vox_dims), 'voxel_dims
↪ '),
                                                                    (('dwi', get_data_dims), 'data_dims
↪ ')])])

```

Finally, we create another higher-level workflow to connect our tractography workflow with the info and data-grabbing nodes declared at the beginning. Our tutorial can is now extensible to any arbitrary number of subjects by simply adding their names to the subject list and their data to the proper folders.

```

workflow = pe.Workflow(name="workflow")
workflow.base_dir = os.path.abspath('camino_dti_tutorial')
workflow.connect([(infosource, datasource, [('subject_id', 'subject_id')],
                (datasource, tractography, [('dwi', 'inputnode.dwi'),
                ('bvals', 'inputnode.bvals'),
                ('bvecs', 'inputnode.bvecs')
                ])
                ])

```

The following functions run the whole workflow and produce a .dot and .png graph of the processing pipeline.

```

if __name__ == '__main__':
    workflow.run()
    workflow.write_graph()

```

You can choose the format of the experted graph with the `format` option. For example `workflow.write_graph(format='eps')`

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

dMRI: Connectivity - Camino, CMTK, FreeSurfer

Introduction

This script, `connectivity_tutorial.py`, demonstrates the ability to perform connectivity mapping using Nipype for pipelining, Freesurfer for Reconstruction / Parcellation, Camino for tensor-fitting and tractography, and the Connectome Mapping Toolkit (CMTK) for connectivity analysis:

```
python connectivity_tutorial.py
```

We perform this analysis using the FSL course data, which can be acquired from here:

- http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

This pipeline also requires the Freesurfer directory for 'subj1' from the FSL course data. To save time, this data can be downloaded from here:

- <http://dl.dropbox.com/u/315714/subj1.zip?dl=1>

A data package containing the outputs of this pipeline can be obtained from here:

- <http://db.tt/1vx4vLeP>

Along with Camino, Camino-Trackvis, FSL, and Freesurfer, you must also have the Connectome File Format library installed as well as the Connectome Mapper.

These are written by Stephan Gerhard and can be obtained from:

<http://www.cmtk.org/>

Or on github at:

CFFlib: <https://github.com/LTS5/cfflib> CMP: <https://github.com/LTS5/cmp>

Output data can be visualized in the ConnectomeViewer

ConnectomeViewer: <https://github.com/LTS5/connectomeviewer>

First, we import the necessary modules from nipype.

```
import inspect

import os.path as op                # system functions
import cmp                          # connectome mapper
import nipype.interfaces.io as nio  # Data i/o
import nipype.interfaces.utility as util  # utility
import nipype.pipeline.engine as pe    # pipeline engine
import nipype.interfaces.camino as camino
import nipype.interfaces.fsl as fsl
import nipype.interfaces.camino2trackvis as cam2trk
import nipype.interfaces.freesurfer as fs  # freesurfer
```

```
import nipy.interfaces.cmtk as cmtk
import nipy.algorithms.misc as misc
```

We define the following functions to scrape the voxel and data dimensions of the input images. This allows the pipeline to be flexible enough to accept and process images of varying size. The SPM Face tutorial (fmri_spm_face.py) also implements this inferral of voxel size from the data. We also define functions to select the proper parcellation/segregation file from Freesurfer's output for each subject. For the mapping in this tutorial, we use the aparc+seg.mgz file. While it is possible to change this to use the regions defined in aparc.a2009s+aseg.mgz, one would also have to write/obtain a network resolution map defining the nodes based on those regions.

```
def get_vox_dims(volume):
    import nibabel as nb
    from nipy.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    voxdims = hdr.get_zooms()
    return [float(voxdims[0]), float(voxdims[1]), float(voxdims[2])]

def get_data_dims(volume):
    import nibabel as nb
    from nipy.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    datadims = hdr.get_data_shape()
    return [int(datadims[0]), int(datadims[1]), int(datadims[2])]

def get_affine(volume):
    import nibabel as nb
    from nipy.utils import NUMPY_MMAP
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    return nii.affine

def select_aparc(list_of_files):
    for in_file in list_of_files:
        if 'aparc+aseg.mgz' in in_file:
            idx = list_of_files.index(in_file)
    return list_of_files[idx]

def select_aparc_annot(list_of_files):
    for in_file in list_of_files:
        if '.aparc.annot' in in_file:
            idx = list_of_files.index(in_file)
    return list_of_files[idx]
```

These need to point to the main Freesurfer directory as well as the freesurfer subjects directory. No assumptions are made about where the directory of subjects is placed. Recon-all must have been run on subj1 from the FSL course data.

```
fs_dir = op.abspath('/usr/local/freesurfer')
subjects_dir = op.abspath(op.join(op.curdir, './subjects'))
fsl.FSLCommand.set_default_output_type('NIFTI')
```

This needs to point to the fdt folder you can find after extracting http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

```
data_dir = op.abspath('fsl_course_data/fdt/')
fs.FSLCommand.set_default_subjects_dir(subjects_dir)
subject_list = ['subj1']
```

An inforsource node is used to loop through the subject list and define the input files. For our purposes, these are the diffusion-weighted MR image, b vectors, and b values. The info dictionary is used to provide a template of the naming of these files. For instance, the 4D nifti diffusion image is stored in the FSL course data as data.nii.gz.

```
infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
    name="infosource")
infosource.iterables = ('subject_id', subject_list)

info = dict(dwi=[['subject_id', 'data']],
            bvecs=[['subject_id', 'bvecs']],
            bvals=[['subject_id', 'bvals']])
```

A datasource node is used to perform the actual data grabbing. Templates for the associated images are used to obtain the correct images. The data are assumed to lie in data_dir/subject_id/.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                                outfields=list(info.keys())),
    name='datasource')

datasource.inputs.template = "%s/%s"
datasource.inputs.base_directory = data_dir
datasource.inputs.field_template = dict(dwi='%s/%s.nii.gz')
datasource.inputs.template_args = info
datasource.inputs.base_directory = data_dir
datasource.inputs.sort_filelist = True
```

FreeSurferSource nodes are used to retrieve a number of image files that were automatically generated by the recon-all process. Here we use three of these nodes, two of which are defined to return files for solely the left and right hemispheres.

```
FreeSurferSource = pe.Node(interface=nio.FreeSurferSource(), name='fssource')
FreeSurferSource.inputs.subjects_dir = subjects_dir

FreeSurferSourceLH = pe.Node(interface=nio.FreeSurferSource(), name='fssourceLH')
FreeSurferSourceLH.inputs.subjects_dir = subjects_dir
FreeSurferSourceLH.inputs.hemi = 'lh'

FreeSurferSourceRH = pe.Node(interface=nio.FreeSurferSource(), name='fssourceRH')
FreeSurferSourceRH.inputs.subjects_dir = subjects_dir
FreeSurferSourceRH.inputs.hemi = 'rh'
```

Since the b values and b vectors come from the FSL course, we must convert it to a scheme file for use in Camino.

```
fsl2scheme = pe.Node(interface=camino.FSL2Scheme(), name="fsl2scheme")
fsl2scheme.inputs.usegradmod = True
```

FSL's Brain Extraction tool is used to create a mask from the b0 image

```
b0Strip = pe.Node(interface=fsl.BET(mask=True), name='bet_b0')
```

FSL's FLIRT function is used to coregister the b0 mask and the structural image. A convert_xfm node is then used to obtain the inverse of the transformation matrix. FLIRT is used once again to apply the inverse transformation to the parcellated brain image.

```
coregister = pe.Node(interface=fsl.FLIRT(dof=6), name='coregister')
coregister.inputs.cost = ('corratio')

convertxfm = pe.Node(interface=fsl.ConvertXFM(), name='convertxfm')
convertxfm.inputs.invert_xfm = True

inverse = pe.Node(interface=fsl.FLIRT(), name='inverse')
inverse.inputs.interp = ('nearestneighbour')

inverse_AparcAseg = pe.Node(interface=fsl.FLIRT(), name='inverse_AparcAseg')
inverse_AparcAseg.inputs.interp = ('nearestneighbour')
```

A number of conversion operations are required to obtain NIFTI files from the FreesurferSource for each subject. Nodes are used to convert the following:

- Original structural image to NIFTI
- Parcellated white matter image to NIFTI
- Parcellated whole-brain image to NIFTI
- **Pial, white, inflated, and spherical surfaces for both the left and right hemispheres** are converted to GIFTI for visualization in ConnectomeViewer
- Parcellated annotation files for the left and right hemispheres are also converted to GIFTI

```
mri_convert_Brain = pe.Node(interface=fs.MRIConvert(), name='mri_convert_Brain')
mri_convert_Brain.inputs.out_type = 'nii'

mri_convert_WMParc = mri_convert_Brain.clone('mri_convert_WMParc')
mri_convert_AparcAseg = mri_convert_Brain.clone('mri_convert_AparcAseg')

mris_convertLH = pe.Node(interface=fs.MRIsConvert(), name='mris_convertLH')
mris_convertLH.inputs.out_datatype = 'gii'
mris_convertRH = mris_convertLH.clone('mris_convertRH')
mris_convertRHwhite = mris_convertLH.clone('mris_convertRHwhite')
mris_convertLHwhite = mris_convertLH.clone('mris_convertLHwhite')
mris_convertRHinflated = mris_convertLH.clone('mris_convertRHinflated')
mris_convertLHinflated = mris_convertLH.clone('mris_convertLHinflated')
mris_convertRHsphere = mris_convertLH.clone('mris_convertRHsphere')
mris_convertLHsphere = mris_convertLH.clone('mris_convertLHsphere')
mris_convertLHlabels = mris_convertLH.clone('mris_convertLHlabels')
mris_convertRHlabels = mris_convertLH.clone('mris_convertRHlabels')
```

An inputnode is used to pass the data obtained by the data grabber to the actual processing functions

```
inputnode = pe.Node(interface=util.IdentityInterface(fields=["dwi", "bvecs",
↪ "bvals", "subject_id"]), name="inputnode")
```

In this section we create the nodes necessary for diffusion analysis. First, the diffusion image is converted to voxel order, since this is the format in which Camino does its processing.

```
image2voxel = pe.Node(interface=camino.Image2Voxel(), name="image2voxel")
```

Second, diffusion tensors are fit to the voxel-order data. If desired, these tensors can be converted to a Nifti tensor image using the DT2Nifti interface.

```
dtifit = pe.Node(interface=camino.DTIFit(), name='dtifit')
```

Next, a lookup table is generated from the schemefile and the signal-to-noise ratio (SNR) of the unweighted (q=0) data.

```
dtlutgen = pe.Node(interface=camino.DTLUTGen(), name="dtlutgen")
dtlutgen.inputs.snr = 16.0
dtlutgen.inputs.inversion = 1
```

In this tutorial we implement probabilistic tractography using the PICO algorithm. PICO tractography requires an estimate of the fibre direction and a model of its uncertainty in each voxel; this probability distribution map is produced using the following node.

```
picopdfs = pe.Node(interface=camino.PicoPDFs(), name="picopdfs")
picopdfs.inputs.inputmodel = 'dt'
```

Finally, tractography is performed. In this tutorial, we will use only one iteration for time-saving purposes. It is important to note that we use the TrackPICO interface here. This interface now expects the files required for PICO tracking (i.e. the output from picopdfs). Similar interfaces exist for alternative types of tracking, such as Bayesian tracking with Dirac priors (TrackBayesDirac).

```
track = pe.Node(interface=camino.TrackPICO(), name="track")
track.inputs.iterations = 1
```

Currently, the best program for visualizing tracts is TrackVis. For this reason, a node is included to convert the raw tract data to .trk format. Solely for testing purposes, another node is added to perform the reverse.

```
camino2trackvis = pe.Node(interface=cam2trk.Camino2Trackvis(), name="camino2trk")
camino2trackvis.inputs.min_length = 30
camino2trackvis.inputs.voxel_order = 'LAS'
trk2camino = pe.Node(interface=cam2trk.Trackvis2Camino(), name="trk2camino")
```

Tracts can also be converted to VTK and OOGL formats, for use in programs such as GeomView and Paraview, using the following two nodes.

```
vtkstreamlines = pe.Node(interface=camino.VtkStreamlines(), name="vtkstreamlines")
procstreamlines = pe.Node(interface=camino.ProcStreamlines(), name=
↳"procstreamlines")
procstreamlines.inputs.outputtracts = 'oogl'
```

We can easily produce a variety of scalar values from our fitted tensors. The following nodes generate the fractional anisotropy and diffusivity trace maps and their associated headers, and then merge them back into a single .nii file.

```
fa = pe.Node(interface=camino.ComputeFractionalAnisotropy(), name='fa')
trace = pe.Node(interface=camino.ComputeTensorTrace(), name='trace')
dteig = pe.Node(interface=camino.ComputeEigensystem(), name='dteig')

analyzeheader_fa = pe.Node(interface=camino.AnalyzeHeader(), name='analyzeheader_
↳fa')
analyzeheader_fa.inputs.datatype = 'double'
analyzeheader_trace = pe.Node(interface=camino.AnalyzeHeader(), name=
↳'analyzeheader_trace')
analyzeheader_trace.inputs.datatype = 'double'

fa2nii = pe.Node(interface=misc.CreateNifti(), name='fa2nii')
trace2nii = fa2nii.clone("trace2nii")
```

This section adds the Connectome Mapping Toolkit (CMTK) nodes. These interfaces are fairly experimental and may not function properly. In order to perform connectivity mapping using CMTK, the parcellated structural data is rewritten using the indices and parcellation scheme from the connectome mapper (CMP). This

process has been written into the ROIgen interface, which will output a remapped aparc+aseg image as well as a dictionary of label information (i.e. name, display colours) pertaining to the original and remapped regions. These label values are input from a user-input lookup table, if specified, and otherwise the default Freesurfer LUT (/freesurfer/FreeSurferColorLUT.txt).

```
roigen = pe.Node(interface=cmtk.ROIgen(), name="ROIgen")
cmp_config = cmp.configuration.PipelineConfiguration(parcellation_scheme=
    ↪ "NativeFreesurfer")
cmp_config.parcellation_scheme = "NativeFreesurfer"
roigen.inputs.LUT_file = cmp_config.get_freeview_lut("NativeFreesurfer") [
    ↪ 'freesurferaparc']
roigen_structspace = roigen.clone('ROIgen_structspace')
```

The CreateMatrix interface takes in the remapped aparc+aseg image as well as the label dictionary and fiber tracts and outputs a number of different files. The most important of which is the connectivity network itself, which is stored as a ‘gpickle’ and can be loaded using Python’s NetworkX package (see CreateMatrix docstring). Also outputted are various NumPy arrays containing detailed tract information, such as the start and endpoint regions, and statistics on the mean and standard deviation for the fiber length of each connection. These matrices can be used in the ConnectomeViewer to plot the specific tracts that connect between user-selected regions.

```
creatematrix = pe.Node(interface=cmtk.CreateMatrix(), name="CreateMatrix")
creatematrix.inputs.count_region_intersections = True
createnodes = pe.Node(interface=cmtk.CreateNodes(), name="CreateNodes")
createnodes.inputs.resolution_network_file = cmp_config.parcellation[
    ↪ 'freesurferaparc'] ['node_information_graphml']
```

Here we define the endpoint of this tutorial, which is the CFFConverter node, as well as a few nodes which use the Nipy Merge utility. These are useful for passing lists of the files we want packaged in our CFF file.

```
CFFConverter = pe.Node(interface=cmtk.CFFConverter(), name="CFFConverter")

giftiSurfaces = pe.Node(interface=util.Merge(8), name="GiftiSurfaces")
giftiLabels = pe.Node(interface=util.Merge(2), name="GiftiLabels")
niftiVolumes = pe.Node(interface=util.Merge(3), name="NiftiVolumes")
fiberDataArrays = pe.Node(interface=util.Merge(4), name="FiberDataArrays")
gpickledNetworks = pe.Node(interface=util.Merge(1), name="NetworkFiles")
```

Since we have now created all our nodes, we can define our workflow and start making connections.

```
mapping = pe.Workflow(name='mapping')
```

First, we connect the input node to the early conversion functions. FreeSurfer input nodes:

```
mapping.connect([(inputnode, FreeSurferSource, [("subject_id", "subject_id")])])
mapping.connect([(inputnode, FreeSurferSourceLH, [("subject_id", "subject_id")])])
mapping.connect([(inputnode, FreeSurferSourceRH, [("subject_id", "subject_id")])])
```

Required conversions for processing in Camino:

```
mapping.connect([(inputnode, image2voxel, [("dwi", "in_file")]),
                (inputnode, fsl2scheme, [("bvecs", "bvec_file"),
                ("bvals", "bval_file")]),
                (image2voxel, dtifit, [['voxel_order', 'in_file']]),
                (fsl2scheme, dtifit, [['scheme', 'scheme_file']])
                ])
```

Nifti conversions for the parcellated white matter image (used in Camino’s conmap), and the subject’s stripped brain image from Freesurfer:

```
mapping.connect([(FreeSurferSource, mri_convert_WMParc, [('wmparc', 'in_file')])])
mapping.connect([(FreeSurferSource, mri_convert_Brain, [('brain', 'in_file')])])
```

Surface conversions to GIFTI (pial, white, inflated, and sphere for both hemispheres)

```
mapping.connect([(FreeSurferSourceLH, mris_convertLH, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRH, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHwhite, [('white', 'in_file'
↪')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHwhite, [('white', 'in_file'
↪')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHinflated, [('inflated', 'in_
↪file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHinflated, [('inflated', 'in_
↪file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHsphere, [('sphere', 'in_file'
↪')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHSphere, [('sphere', 'in_file'
↪')])])
```

The annotation files are converted using the pial surface as a map via the MRIsConvert interface. One of the functions defined earlier is used to select the lh.aparc.annot and rh.aparc.annot files specifically (rather than i.e. rh.aparc.a2009s.annot) from the output list given by the FreeSurferSource.

```
mapping.connect([(FreeSurferSourceLH, mris_convertLHlabels, [('pial', 'in_file'
↪')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHlabels, [('pial', 'in_file'
↪')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHlabels, [(['annot', select_
↪aparc_annot), 'annot_file'])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHlabels, [(['annot', select_
↪aparc_annot), 'annot_file'])])
```

This section coregisters the diffusion-weighted and parcellated white-matter / whole brain images. At present the conmap node connection is left commented, as there have been recent changes in Camino code that have presented some users with errors.

```
mapping.connect([(inputnode, b0Strip, [('dwi', 'in_file')])])
mapping.connect([(b0Strip, coregister, [('out_file', 'in_file')])])
mapping.connect([(mri_convert_Brain, coregister, [('out_file', 'reference')])])
mapping.connect([(coregister, convertxfm, [('out_matrix_file', 'in_file')])])
mapping.connect([(b0Strip, inverse, [('out_file', 'reference')])])
mapping.connect([(convertxfm, inverse, [('out_file', 'in_matrix_file')])])
mapping.connect([(mri_convert_WMParc, inverse, [('out_file', 'in_file')])])
```

The tractography pipeline consists of the following nodes. Further information about the tractography can be found in `nipy/examples/dmri_camino_dti.py`.

```
mapping.connect([(b0Strip, track, [("mask_file", "seed_file")])])
mapping.connect([(fsl2scheme, dtlutgen, [("scheme", "scheme_file")])])
mapping.connect([(dtlutgen, picopdfs, [("dtLUT", "luts")])])
mapping.connect([(dtifit, picopdfs, [("tensor_fitted", "in_file")])])
mapping.connect([(picopdfs, track, [("pdfs", "in_file")])])
```

Connecting the Fractional Anisotropy and Trace nodes is simple, as they obtain their input from the tensor fitting. This is also where our voxel- and data-grabbing functions come in. We pass these functions, along with the original DWI image from the input node, to the header-generating nodes. This ensures that the files will be correct and readable.

```
mapping.connect([(dtifit, fa, [("tensor_fitted", "in_file")])])
mapping.connect([(fa, analyzeheader_fa, [("fa", "in_file")])])
mapping.connect([(inputnode, analyzeheader_fa, [(['dwi', get_vox_dims), 'voxel_
↪dims']),
```

```

                                (('dwi', get_data_dims), 'data_
↪dims'))]]))
mapping.connect([(fa, fa2nii, [('fa', 'data_file')])])
mapping.connect([(inputnode, fa2nii, [('dwi', get_affine), 'affine'])])
mapping.connect([(analyzeheader_fa, fa2nii, [('header', 'header_file')])])

mapping.connect([(dtifit, trace, [("tensor_fitted", "in_file")])])
mapping.connect([(trace, analyzeheader_trace, [("trace", "in_file")])])
mapping.connect([(inputnode, analyzeheader_trace, [('dwi', get_vox_dims), 'voxel_
↪dims'],
                                (('dwi', get_data_dims), 'data_
↪dims'))]]))
mapping.connect([(trace, trace2nii, [('trace', 'data_file')])])
mapping.connect([(inputnode, trace2nii, [('dwi', get_affine), 'affine'])])
mapping.connect([(analyzeheader_trace, trace2nii, [('header', 'header_file')])])

mapping.connect([(dtifit, dteig, [("tensor_fitted", "in_file")])])

```

The output tracts are converted to Trackvis format (and back). Here we also use the voxel- and data-grabbing functions defined at the beginning of the pipeline.

```

mapping.connect([(track, camino2trackvis, [('tracked', 'in_file')]),
                (track, vtkstreamlines, [('tracked', 'in_file')]),
                (camino2trackvis, trk2camino, [('trackvis', 'in_file')])
                ])
mapping.connect([(inputnode, camino2trackvis, [('dwi', get_vox_dims), 'voxel_dims
↪'],
                                (('dwi', get_data_dims), 'data_dims
↪'))]])

```

Here the CMTK connectivity mapping nodes are connected. The original aparc+aseg image is converted to NIFTI, then registered to the diffusion image and delivered to the ROIgen node. The remapped parcellation, original tracts, and label file are then given to CreateMatrix.

```

mapping.connect(createnodes, 'node_network',
                creatematrix, 'resolution_network_file')
mapping.connect([(FreeSurferSource, mri_convert_AparcAseg, [('aparc_aseg',
↪select_aparc), 'in_file'])])

mapping.connect([(b0Strip, inverse_AparcAseg, [('out_file', 'reference')])])
mapping.connect([(convertxfm, inverse_AparcAseg, [('out_file', 'in_matrix_file
↪')])])
mapping.connect([(mri_convert_AparcAseg, inverse_AparcAseg, [('out_file', 'in_file
↪')])])
mapping.connect([(mri_convert_AparcAseg, roigen_structspace, [('out_file', 'aparc_
↪aseg_file')])])
mapping.connect([(roigen_structspace, createnodes, [("roi_file", "roi_file")])])

mapping.connect([(inverse_AparcAseg, roigen, [("out_file", "aparc_aseg_file")])])
mapping.connect([(roigen, creatematrix, [("roi_file", "roi_file")])])
mapping.connect([(camino2trackvis, creatematrix, [("trackvis", "tract_file")])])
mapping.connect([(inputnode, creatematrix, [("subject_id", "out_matrix_file")])])
mapping.connect([(inputnode, creatematrix, [("subject_id", "out_matrix_mat_file
↪")])])

```

The merge nodes defined earlier are used here to create lists of the files which are destined for the CFFConverter.


```

mapping.connect([(creatematrix, gpickledNetworks, [("matrix_files", "in1")])])

mapping.connect([(mris_convertLH, giftiSurfaces, [("converted", "in1")])])
mapping.connect([(mris_convertRH, giftiSurfaces, [("converted", "in2")])])
mapping.connect([(mris_convertLHwhite, giftiSurfaces, [("converted", "in3")])])
mapping.connect([(mris_convertRHwhite, giftiSurfaces, [("converted", "in4")])])
mapping.connect([(mris_convertLHinflated, giftiSurfaces, [("converted", "in5")])])
mapping.connect([(mris_convertRHinflated, giftiSurfaces, [("converted", "in6")])])
mapping.connect([(mris_convertLHsphere, giftiSurfaces, [("converted", "in7")])])
mapping.connect([(mris_convertRHSphere, giftiSurfaces, [("converted", "in8")])])

mapping.connect([(mris_convertLHlabels, giftiLabels, [("converted", "in1")])])
mapping.connect([(mris_convertRHlabels, giftiLabels, [("converted", "in2")])])

mapping.connect([(roigen, niftiVolumes, [("roi_file", "in1")])])
mapping.connect([(inputnode, niftiVolumes, [("dwi", "in2")])])
mapping.connect([(mri_convert_Brain, niftiVolumes, [("out_file", "in3")])])

mapping.connect([(creatematrix, fiberDataArrays, [("endpoint_file", "in1")])])
mapping.connect([(creatematrix, fiberDataArrays, [("endpoint_file_mm", "in2")])])
mapping.connect([(creatematrix, fiberDataArrays, [("fiber_length_file", "in3")])])
mapping.connect([(creatematrix, fiberDataArrays, [("fiber_label_file", "in4")])])

```

This block actually connects the merged lists to the CFF converter. We pass the surfaces and volumes that are to be included, as well as the tracts and the network itself. The currently running pipeline (`dmri_connectivity.py`) is also scraped and included in the CFF file. This makes it easy for the user to examine the entire processing pathway used to generate the end product.

```

CFFConverter.inputs.script_files = op.abspath(inspect.getfile(inspect.
↪currentframe()))
mapping.connect([(giftiSurfaces, CFFConverter, [("out", "gifti_surfaces")])])
mapping.connect([(giftiLabels, CFFConverter, [("out", "gifti_labels")])])
mapping.connect([(gpickledNetworks, CFFConverter, [("out", "gpickled_networks
↪")])])
mapping.connect([(niftiVolumes, CFFConverter, [("out", "nifti_volumes")])])
mapping.connect([(fiberDataArrays, CFFConverter, [("out", "data_files")])])
mapping.connect([(creatematrix, CFFConverter, [("filtered_tractographies", "tract_
↪files")])])
mapping.connect([(inputnode, CFFConverter, [("subject_id", "title")])])

```

Finally, we create another higher-level workflow to connect our mapping workflow with the info and datagraphing nodes declared at the beginning. Our tutorial can is now extensible to any arbitrary number of subjects by simply adding their names to the subject list and their data to the proper folders.

```

connectivity = pe.Workflow(name="connectivity")
connectivity.base_dir = op.abspath('dmri_connectivity')
connectivity.connect([
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (datasource, mapping, [('dwi', 'inputnode.dwi'),
                           ('bvals', 'inputnode.bvals'),
                           ('bvecs', 'inputnode.bvecs')
                           ]),
    (infosource, mapping, [('subject_id', 'inputnode.subject_id')])
])

```

The following functions run the whole workflow and produce graphs describing the processing pipeline. By default, `write_graph` outputs a `.dot` file and a `.png` image, but here we set it to output the image as a vector graphic, by passing the `format='eps'` argument.

```
if __name__ == '__main__':
    connectivity.run()
    connectivity.write_graph(format='eps')
```

The output CFF file of this pipeline can be loaded in the [Connectome Viewer](#). After loading the network into memory it can be examined in 3D or as a connectivity matrix using the default scripts produced by the Code Oracle. To compare networks, one must use the MergeCNetworks interface to merge two networks into a single CFF file. Statistics can then be run using the Network Brain Statistics (NBS) plugin Surfaces can also be loaded along with their labels from the aparc+aseg file. The tractography is included in the file so that region-to-region fibers can be individually plotted using the Code Oracle.

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

dMRI: Connectivity - MRtrix, CMTK, FreeSurfer

Introduction

This script, `connectivity_tutorial_advanced.py`, demonstrates the ability to perform connectivity mapping using Nipype for pipelining, Freesurfer for Reconstruction / Segmentation, MRtrix for spherical deconvolution and tractography, and the Connectome Mapping Toolkit (CMTK) for further parcellation and connectivity analysis:

```
python connectivity_tutorial_advanced.py
```

We perform this analysis using the FSL course data, which can be acquired from here:

- http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

This pipeline also requires the Freesurfer directory for 'subj1' from the FSL course data. To save time, this data can be downloaded from here:

- <http://dl.dropbox.com/u/315714/subj1.zip?dl=1>

The result of this processing will be the connectome for subj1 as a Connectome File Format (CFF) File, using the Lausanne2008 parcellation scheme. A data package containing the outputs of this pipeline can be obtained from here:

- <http://db.tt/909Q3AC1>

See also:

connectivity_tutorial.py Original tutorial using Camino and the NativeFreesurfer Parcellation Scheme

www.cmtk.org For more info about the parcellation scheme

Warning: The ConnectomeMapper (<https://github.com/LTS5/cmp> or www.cmtk.org) must be installed for this tutorial to function!

Packages and Data Setup

Import necessary modules from nipype.

```
import nipype.interfaces.io as nio           # Data i/o
import nipype.interfaces.utility as util      # utility
import nipype.pipeline.engine as pe          # pipeline engine
import nipype.interfaces.fsl as fsl
import nipype.interfaces.freesurfer as fs     # freesurfer
import nipype.interfaces.mrtrix as mrtrix
```

```

import nipyype.algorithms.misc as misc
import nipyype.interfaces.cmtk as cmtk
import nipyype.interfaces.dipy as dipy
import inspect
import os
import os.path as op                                # system functions
from nipyype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline
from nipyype.workflows.dmri.camino.connectivity_mapping import select_aparc_annot
from nipyype.utils.misc import package_check
import warnings
from nipyype.workflows.dmri.connectivity.nx import create_networkx_pipeline, \
↳ create_cmats_to_csv_pipeline
from nipyype.workflows.smri.freesurfer import create_tessellation_flow

try:
    package_check('cmp')
except Exception as e:
    warnings.warn('cmp not installed')
else:
    import cmp

```

This needs to point to the freesurfer subjects directory (Recon-all must have been run on subj1 from the FSL course data) Alternatively, the reconstructed subject data can be downloaded from:

- <http://dl.dropbox.com/u/315714/subj1.zip>

```

subjects_dir = op.abspath(op.join(op.curdir, './subjects'))
fs.FSCommand.set_default_subjects_dir(subjects_dir)
fsl.FSLCommand.set_default_output_type('NIFTI')

fs_dir = os.environ['FREESURFER_HOME']
lookup_file = op.join(fs_dir, 'FreeSurferColorLUT.txt')

```

This needs to point to the fdt folder you can find after extracting

- http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

```

data_dir = op.abspath(op.join(op.curdir, 'exdata/'))
subject_list = ['subj1']

```

Use infosource node to loop through the subject list and define the input files. For our purposes, these are the diffusion-weighted MR image, b vectors, and b values.

```

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']), \
↳ name="infosource")
infosource.iterables = ('subject_id', subject_list)

info = dict(dwi=[['subject_id', 'data']],
            bvecs=[['subject_id', 'bvecs']],
            bvals=[['subject_id', 'bvals']])

```

Use datasource node to perform the actual data grabbing. Templates for the associated images are used to obtain the correct images.

```

datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                                outfields=list(info.keys())),
                    name='datasource')

datasource.inputs.template = "%s/%s"
datasource.inputs.base_directory = data_dir
datasource.inputs.field_template = dict(dwi='%s/%s.nii.gz')
datasource.inputs.template_args = info

```

```
datasource.inputs.sort_filelist = True
```

The input node and Freesurfer sources declared here will be the main conduits for the raw data to the rest of the processing pipeline.

```
inputnode = pe.Node(interface=util.IdentityInterface(fields=["subject_id", "dwi",
↳ "bvecs", "bvals", "subjects_dir"]), name="inputnode")
inputnode.inputs.subjects_dir = subjects_dir

FreeSurferSource = pe.Node(interface=nio.FreeSurferSource(), name='fssource')
FreeSurferSourceLH = FreeSurferSource.clone('fssourceLH')
FreeSurferSourceLH.inputs.hemi = 'lh'
FreeSurferSourceRH = FreeSurferSource.clone('fssourceRH')
FreeSurferSourceRH.inputs.hemi = 'rh'
```

Creating the workflow's nodes

Conversion nodes

A number of conversion operations are required to obtain NIFTI files from the FreesurferSource for each subject. Nodes are used to convert the following:

- Original structural image to NIFTI
- Pial, white, inflated, and spherical surfaces for both the left and right hemispheres are converted to GIFTI for visualization in ConnectomeViewer
- Parcellated annotation files for the left and right hemispheres are also converted to GIFTI

```
mri_convert_Brain = pe.Node(interface=fs.MRIConvert(), name='mri_convert_Brain')
mri_convert_Brain.inputs.out_type = 'nii'
mri_convert_ROI_scale500 = mri_convert_Brain.clone('mri_convert_ROI_scale500')

mris_convertLH = pe.Node(interface=fs.MRIsConvert(), name='mris_convertLH')
mris_convertLH.inputs.out_datatype = 'gii'
mris_convertRH = mris_convertLH.clone('mris_convertRH')
mris_convertRHwhite = mris_convertLH.clone('mris_convertRHwhite')
mris_convertLHwhite = mris_convertLH.clone('mris_convertLHwhite')
mris_convertRHinflated = mris_convertLH.clone('mris_convertRHinflated')
mris_convertLHinflated = mris_convertLH.clone('mris_convertLHinflated')
mris_convertRHsphere = mris_convertLH.clone('mris_convertRHsphere')
mris_convertLHsphere = mris_convertLH.clone('mris_convertLHsphere')
mris_convertLHlabels = mris_convertLH.clone('mris_convertLHlabels')
mris_convertRHlabels = mris_convertLH.clone('mris_convertRHlabels')
```

Diffusion processing nodes

See also:

dmri_mrtrix_dti.py Tutorial that focuses solely on the MRtrix diffusion processing

<http://www.brain.org.au/software/mrtrix/index.html> MRtrix's online documentation

b-values and b-vectors stored in FSL's format are converted into a single encoding file for MRTrix.

```
fsl2mrtrix = pe.Node(interface=mrtrix.FSL2MRTrix(), name='fsl2mrtrix')
```

Distortions induced by eddy currents are corrected prior to fitting the tensors. The first image is used as a reference for which to warp the others.

```
eddycorrect = create_eddy_correct_pipeline(name='eddycorrect')
eddycorrect.inputs.inputnode.ref_num = 1
```

Tensors are fitted to each voxel in the diffusion-weighted image and from these three maps are created:

- Major eigenvector in each voxel
- Apparent diffusion coefficient
- Fractional anisotropy

```
dwi2tensor = pe.Node(interface=mrtrix.DWI2Tensor(), name='dwi2tensor')
tensor2vector = pe.Node(interface=mrtrix.Tensor2Vector(), name='tensor2vector')
tensor2adc = pe.Node(interface=mrtrix.Tensor2ApparentDiffusion(), name='tensor2adc
↳')
tensor2fa = pe.Node(interface=mrtrix.Tensor2FractionalAnisotropy(), name=
↳'tensor2fa')
MRconvert_fa = pe.Node(interface=mrtrix.MRConvert(), name='MRconvert_fa')
MRconvert_fa.inputs.extension = 'nii'
```

These nodes are used to create a rough brain mask from the b0 image. The b0 image is extracted from the original diffusion-weighted image, put through a simple thresholding routine, and smoothed using a 3x3 median filter.

```
MRconvert = pe.Node(interface=mrtrix.MRConvert(), name='MRconvert')
MRconvert.inputs.extract_at_axis = 3
MRconvert.inputs.extract_at_coordinate = [0]
threshold_b0 = pe.Node(interface=mrtrix.Threshold(), name='threshold_b0')
median3d = pe.Node(interface=mrtrix.MedianFilter3D(), name='median3d')
```

The brain mask is also used to help identify single-fiber voxels. This is done by passing the brain mask through two erosion steps, multiplying the remaining mask with the fractional anisotropy map, and thresholding the result to obtain some highly anisotropic within-brain voxels.

```
erode_mask_firstpass = pe.Node(interface=mrtrix.Erode(), name='erode_mask_
↳firstpass')
erode_mask_secondpass = pe.Node(interface=mrtrix.Erode(), name='erode_mask_
↳secondpass')
MRmultiply = pe.Node(interface=mrtrix.MRMultiply(), name='MRmultiply')
MRmult_merge = pe.Node(interface=util.Merge(2), name='MRmultiply_merge')
threshold_FA = pe.Node(interface=mrtrix.Threshold(), name='threshold_FA')
threshold_FA.inputs.absolute_threshold_value = 0.7
```

For whole-brain tracking we also require a broad white-matter seed mask. This is created by generating a white matter mask, given a brainmask, and thresholding it at a reasonably high level.

```
bet = pe.Node(interface=fsl.BET(mask=True), name='bet_b0')
gen_WM_mask = pe.Node(interface=mrtrix.GenerateWhiteMatterMask(), name='gen_WM_
↳mask')
threshold_wmmask = pe.Node(interface=mrtrix.Threshold(), name='threshold_wmmask')
threshold_wmmask.inputs.absolute_threshold_value = 0.4
```

The spherical deconvolution step depends on the estimate of the response function in the highly anisotropic voxels we obtained above.

Warning: For damaged or pathological brains one should take care to lower the maximum harmonic order of these steps.

```
estimatorresponse = pe.Node(interface=mrtrix.EstimateResponseForSH(), name=
↳'estimatorresponse')
estimatorresponse.inputs.maximum_harmonic_order = 6
csdeconv = pe.Node(interface=mrtrix.ConstrainedSphericalDeconvolution(), name=
↳'csdeconv')
csdeconv.inputs.maximum_harmonic_order = 6
```

Finally, we track probabilistically using the orientation distribution functions obtained earlier. The tracts are then used to generate a tract-density image, and they are also converted to TrackVis format.

```
probCSDstreamtrack = pe.Node(interface=mrtrix.
    ↳ ProbabilisticSphericallyDeconvolutedStreamlineTrack(), name='probCSDstreamtrack'
    ↳ '')
probCSDstreamtrack.inputs.inputmodel = 'SD_PROB'
probCSDstreamtrack.inputs.desired_number_of_tracks = 150000
tracks2prob = pe.Node(interface=mrtrix.Tracks2Prob(), name='tracks2prob')
tracks2prob.inputs.colour = True
MRconvert_tracks2prob = MRconvert_fa.clone(name='MRconvert_tracks2prob')
tck2trk = pe.Node(interface=mrtrix.MRTrix2TrackVis(), name='tck2trk')
trk2tdi = pe.Node(interface=dipy.TrackDensityMap(), name='trk2tdi')
```

Structural segmentation nodes

The following node identifies the transformation between the diffusion-weighted image and the structural image. This transformation is then applied to the tracts so that they are in the same space as the regions of interest.

```
coregister = pe.Node(interface=fsl.FLIRT(dof=6), name='coregister')
coregister.inputs.cost = ('normmi')
```

Parcellation is performed given the aparc+aseg image from Freesurfer. The CMTK Parcellation step subdivides these regions to return a higher-resolution parcellation scheme. The parcellation used here is entitled “scale500” and returns 1015 regions.

```
parcellation_name = 'scale500'
parcellate = pe.Node(interface=cmtk.Parcellate(), name="Parcellate")
parcellate.inputs.parcellation_name = parcellation_name
```

The CreateMatrix interface takes in the remapped aparc+aseg image as well as the label dictionary and fiber tracts and outputs a number of different files. The most important of which is the connectivity network itself, which is stored as a ‘gpickle’ and can be loaded using Python’s NetworkX package (see CreateMatrix docstring). Also outputted are various NumPy arrays containing detailed tract information, such as the start and endpoint regions, and statistics on the mean and standard deviation for the fiber length of each connection. These matrices can be used in the ConnectomeViewer to plot the specific tracts that connect between user-selected regions. Here we choose the Lausanne2008 parcellation scheme, since we are incorporating the CMTK parcellation step.

```
parcellation_name = 'scale500'
cmp_config = cmp.configuration.PipelineConfiguration()
cmp_config.parcellation_scheme = "Lausanne2008"
createnodes = pe.Node(interface=cmtk.CreateNodes(), name="CreateNodes")
createnodes.inputs.resolution_network_file = cmp_config._get_lausanne_
    ↳ parcellation('Lausanne2008')[parcellation_name]['node_information_graphml']

creatematrix = pe.Node(interface=cmtk.CreateMatrix(), name="CreateMatrix")
creatematrix.inputs.count_region_intersections = True
```

Next we define the endpoint of this tutorial, which is the CFFConverter node, as well as a few nodes which use the Nipype Merge utility. These are useful for passing lists of the files we want packaged in our CFF file. The inspect.getfile command is used to package this script into the resulting CFF file, so that it is easy to look back at the processing parameters that were used.

```
CFFConverter = pe.Node(interface=cmtk.CFFConverter(), name="CFFConverter")
CFFConverter.inputs.script_files = op.abspath(inspect.getfile(inspect.
    ↳ currentframe()))
giftiSurfaces = pe.Node(interface=util.Merge(9), name="GiftiSurfaces")
giftiLabels = pe.Node(interface=util.Merge(2), name="GiftiLabels")
```

```
niftiVolumes = pe.Node(interface=util.Merge(3), name="NiftiVolumes")
fiberDataArrays = pe.Node(interface=util.Merge(4), name="FiberDataArrays")
gpickledNetworks = pe.Node(interface=util.Merge(2), name="NetworkFiles")
```

We also create a workflow to calculate several network metrics on our resulting file, and another CFF converter which will be used to package these networks into a single file.

```
networkx = create_networkx_pipeline(name='networkx')
cmats_to_csv = create_cmats_to_csv_pipeline(name='cmats_to_csv')
NxStatsCFFConverter = pe.Node(interface=cmtk.CFFConverter(), name=
    ↪ "NxStatsCFFConverter")
NxStatsCFFConverter.inputs.script_files = op.abspath(inspect.getfile(inspect.
    ↪ currentframe()))

tessflow = create_tessellation_flow(name='tessflow', out_format='gii')
tessflow.inputs.inputs.spec.lookup_file = lookup_file
```

Connecting the workflow

Here we connect our processing pipeline.

Connecting the inputs, FreeSurfer nodes, and conversions

```
mapping = pe.Workflow(name='mapping')
```

First, we connect the input node to the FreeSurfer input nodes.

```
mapping.connect([(inputnode, FreeSurferSource, [("subjects_dir", "subjects_dir"
    ↪)])])
mapping.connect([(inputnode, FreeSurferSource, [("subject_id", "subject_id")])])

mapping.connect([(inputnode, FreeSurferSourceLH, [("subjects_dir", "subjects_dir"
    ↪)])])
mapping.connect([(inputnode, FreeSurferSourceLH, [("subject_id", "subject_id")])])

mapping.connect([(inputnode, FreeSurferSourceRH, [("subjects_dir", "subjects_dir"
    ↪)])])
mapping.connect([(inputnode, FreeSurferSourceRH, [("subject_id", "subject_id")])])

mapping.connect([(inputnode, tessflow, [("subjects_dir", "inputs.spec.subjects_dir"
    ↪)])])
mapping.connect([(inputnode, tessflow, [("subject_id", "inputs.spec.subject_id")])])

mapping.connect([(inputnode, parcellate, [("subjects_dir", "subjects_dir")])])
mapping.connect([(inputnode, parcellate, [("subject_id", "subject_id")])])
mapping.connect([(parcellate, mri_convert_ROI_scale500, [('roi_file', 'in_file'
    ↪)])])
```

Nifti conversion for subject's stripped brain image from FreeSurfer:

```
mapping.connect([(FreeSurferSource, mri_convert_Brain, [('brain', 'in_file')])])
```

Surface conversions to GIFTI (pial, white, inflated, and sphere for both hemispheres)

```
mapping.connect([(FreeSurferSourceLH, mris_convertLH, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRH, [('pial', 'in_file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHwhite, [('white', 'in_file'
    ↪)])])
```



```
mapping.connect([(FreeSurferSourceRH, mris_convertRHwhite, [('white', 'in_file
↪')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHinflated, [('inflated', 'in
↪file')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHinflated, [('inflated', 'in
↪file')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHsphere, [('sphere', 'in_file
↪')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHSphere, [('sphere', 'in_file
↪')])])
```

The annotation files are converted using the pial surface as a map via the MRIsConvert interface. One of the functions defined earlier is used to select the lh.aparc.annot and rh.aparc.annot files specifically (rather than e.g. rh.aparc.a2009s.annot) from the output list given by the FreeSurferSource.

```
mapping.connect([(FreeSurferSourceLH, mris_convertLHlabels, [('pial', 'in_file
↪')])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHlabels, [('pial', 'in_file
↪')])])
mapping.connect([(FreeSurferSourceLH, mris_convertLHlabels, [('annot', select_
↪aparc_annot), 'annot_file'])])
mapping.connect([(FreeSurferSourceRH, mris_convertRHlabels, [('annot', select_
↪aparc_annot), 'annot_file'])])
```

Diffusion Processing

Now we connect the tensor computations:

```
mapping.connect([(inputnode, fsl2mrtrix, [("bvecs", "bvec_file"),
                                          ("bvals", "bval_file")])])
mapping.connect([(inputnode, eddycorrect, [("dwi", "inputnode.in_file")])])
mapping.connect([(eddycorrect, dwi2tensor, [("outputnode.eddy_corrected", "in_file
↪")])])
mapping.connect([(fsl2mrtrix, dwi2tensor, [("encoding_file", "encoding_file")])])

mapping.connect([(dwi2tensor, tensor2vector, [{"tensor", "in_file"}]),
                 (dwi2tensor, tensor2adc, [{"tensor", "in_file"}]),
                 (dwi2tensor, tensor2fa, [{"tensor", "in_file"}]),
                 ])
mapping.connect([(tensor2fa, MRmult_merge, [("FA", "in1")])])
mapping.connect([(tensor2fa, MRconvert_fa, [("FA", "in_file")])])
```

This block creates the rough brain mask to be multiplied, multiplies it with the fractional anisotropy image, and thresholds it to get the single-fiber voxels.

```
mapping.connect([(eddycorrect, MRconvert, [("outputnode.eddy_corrected", "in_file
↪")])])
mapping.connect([(MRconvert, threshold_b0, [("converted", "in_file")])])
mapping.connect([(threshold_b0, median3d, [("out_file", "in_file")])])
mapping.connect([(median3d, erode_mask_firstpass, [("out_file", "in_file")])])
mapping.connect([(erode_mask_firstpass, erode_mask_secondpass, [("out_file", "in
↪file")])])
mapping.connect([(erode_mask_secondpass, MRmult_merge, [("out_file", "in2")])])
mapping.connect([(MRmult_merge, MRmultiply, [("out", "in_files")])])
mapping.connect([(MRmultiply, threshold_FA, [("out_file", "in_file")])])
```

Here the thresholded white matter mask is created for seeding the tractography.

```
mapping.connect([(eddycorrect, bet, [("outputnode.eddy_corrected", "in_file")])])
mapping.connect([(eddycorrect, gen_WM_mask, [("outputnode.eddy_corrected", "in_
↪file")])])
mapping.connect([(bet, gen_WM_mask, [("mask_file", "binary_mask")])])
mapping.connect([(fsl2mrtrix, gen_WM_mask, [("encoding_file", "encoding_file")])])
mapping.connect([(gen_WM_mask, threshold_wmmask, [("WMprobabilitymap", "in_file
↪")])])
```

Next we estimate the fiber response distribution.

```
mapping.connect([(eddycorrect, estimatorresponse, [("outputnode.eddy_corrected",
↪"in_file")])])
mapping.connect([(fsl2mrtrix, estimatorresponse, [("encoding_file", "encoding_file
↪")])])
mapping.connect([(threshold_FA, estimatorresponse, [("out_file", "mask_image")])])
```

Run constrained spherical deconvolution.

```
mapping.connect([(eddycorrect, csdeconv, [("outputnode.eddy_corrected", "in_file
↪")])])
mapping.connect([(gen_WM_mask, csdeconv, [("WMprobabilitymap", "mask_image")])])
mapping.connect([(estimatorresponse, csdeconv, [("response", "response_file")])])
mapping.connect([(fsl2mrtrix, csdeconv, [("encoding_file", "encoding_file")])])
```

Connect the tractography and compute the tract density image.

```
mapping.connect([(threshold_wmmask, probCSDstreamtrack, [("out_file", "seed_file
↪")])])
mapping.connect([(csdeconv, probCSDstreamtrack, [("spherical_harmonics_image",
↪"in_file")])])
mapping.connect([(probCSDstreamtrack, tracks2prob, [("tracked", "in_file")])])
mapping.connect([(eddycorrect, tracks2prob, [("outputnode.eddy_corrected",
↪"template_file")])])
mapping.connect([(tracks2prob, MRconvert_tracks2prob, [("tract_image", "in_file
↪")])])
```

Structural Processing

First, we coregister the diffusion image to the structural image

```
mapping.connect([(eddycorrect, coregister, [("outputnode.eddy_corrected", "in_file
↪")])])
mapping.connect([(mri_convert_Brain, coregister, [('out_file', 'reference')])])
```

The MRtrix-tracked fibers are converted to TrackVis format (with voxel and data dimensions grabbed from the DWI). The connectivity matrix is created with the transformed .trk fibers and the parcellation file.

```
mapping.connect([(eddycorrect, tck2trk, [("outputnode.eddy_corrected", "image_file
↪")])])
mapping.connect([(mri_convert_Brain, tck2trk, [("out_file", "registration_image_
↪file")])])
mapping.connect([(coregister, tck2trk, [("out_matrix_file", "matrix_file")])])
mapping.connect([(probCSDstreamtrack, tck2trk, [("tracked", "in_file")])])
mapping.connect([(tck2trk, creatematrix, [("out_file", "tract_file")])])
mapping.connect([(tck2trk, trk2tdi, [("out_file", "in_file")])])
mapping.connect([(inputnode, creatematrix, [("subject_id", "out_matrix_file")])])
mapping.connect([(inputnode, creatematrix, [("subject_id", "out_matrix_mat_file
↪")])])
mapping.connect([(parcellate, creatematrix, [("roi_file", "roi_file")])])
```

```
mapping.connect([(parcellate, createnodes, [("roi_file", "roi_file")])])
mapping.connect([(createnodes, creatematrix, [("node_network", "resolution_
↪network_file")])])
```

The merge nodes defined earlier are used here to create lists of the files which are destined for the CFFConverter.

```
mapping.connect([(mris_convertLH, giftiSurfaces, [("converted", "in1")])])
mapping.connect([(mris_convertRH, giftiSurfaces, [("converted", "in2")])])
mapping.connect([(mris_convertLHwhite, giftiSurfaces, [("converted", "in3")])])
mapping.connect([(mris_convertRHwhite, giftiSurfaces, [("converted", "in4")])])
mapping.connect([(mris_convertLHinflated, giftiSurfaces, [("converted", "in5")])])
mapping.connect([(mris_convertRHinflated, giftiSurfaces, [("converted", "in6")])])
mapping.connect([(mris_convertLHsphere, giftiSurfaces, [("converted", "in7")])])
mapping.connect([(mris_convertRHsphere, giftiSurfaces, [("converted", "in8")])])
mapping.connect([(tessflow, giftiSurfaces, [("outputspec.meshes", "in9")])])

mapping.connect([(mris_convertLHlabels, giftiLabels, [("converted", "in1")])])
mapping.connect([(mris_convertRHlabels, giftiLabels, [("converted", "in2")])])

mapping.connect([(parcellate, niftiVolumes, [("roi_file", "in1")])])
mapping.connect([(eddycorrect, niftiVolumes, [("outputnode.eddy_corrected", "in2
↪")])])
mapping.connect([(mri_convert_Brain, niftiVolumes, [("out_file", "in3")])])

mapping.connect([(creatematrix, fiberDataArrays, [("endpoint_file", "in1")])])
mapping.connect([(creatematrix, fiberDataArrays, [("endpoint_file_mm", "in2")])])
mapping.connect([(creatematrix, fiberDataArrays, [("fiber_length_file", "in3")])])
mapping.connect([(creatematrix, fiberDataArrays, [("fiber_label_file", "in4")])])
```

This block actually connects the merged lists to the CFF converter. We pass the surfaces and volumes that are to be included, as well as the tracts and the network itself. The currently running pipeline (dmri_connectivity_advanced.py) is also scraped and included in the CFF file. This makes it easy for the user to examine the entire processing pathway used to generate the end product.

```
mapping.connect([(giftiSurfaces, CFFConverter, [("out", "gifti_surfaces")])])
mapping.connect([(giftiLabels, CFFConverter, [("out", "gifti_labels")])])
mapping.connect([(creatematrix, CFFConverter, [("matrix_files", "gpickled_networks
↪")])])
mapping.connect([(niftiVolumes, CFFConverter, [("out", "nifti_volumes")])])
mapping.connect([(fiberDataArrays, CFFConverter, [("out", "data_files")])])
mapping.connect([(creatematrix, CFFConverter, [("filtered_tractographies", "tract_
↪files")])])
mapping.connect([(inputnode, CFFConverter, [("subject_id", "title")])])
```

The graph theoretical metrics are computed using the networkx workflow and placed in another CFF file

```
mapping.connect([(inputnode, networkx, [("subject_id", "inputnode.extra_field
↪")])])
mapping.connect([(creatematrix, networkx, [("intersection_matrix_file",
↪"inputnode.network_file")])])

mapping.connect([(networkx, NxStatsCFFConverter, [("outputnode.network_files",
↪"gpickled_networks")])])
mapping.connect([(giftiSurfaces, NxStatsCFFConverter, [("out", "gifti_surfaces
↪")])])
mapping.connect([(giftiLabels, NxStatsCFFConverter, [("out", "gifti_labels")])])
mapping.connect([(niftiVolumes, NxStatsCFFConverter, [("out", "nifti_volumes")])])
mapping.connect([(fiberDataArrays, NxStatsCFFConverter, [("out", "data_files")])])
mapping.connect([(inputnode, NxStatsCFFConverter, [("subject_id", "title")])])
```

```
mapping.connect([(inputnode, cmats_to_csv, [("subject_id", "inputnode.extra_field
↪")])])
mapping.connect([(creatematrix, cmats_to_csv, [("matlab_matrix_files", "inputnode.
↪matlab_matrix_files")])])
```

Create a higher-level workflow

Finally, we create another higher-level workflow to connect our mapping workflow with the info and datagrabbing nodes declared at the beginning. Our tutorial is now extensible to any arbitrary number of subjects by simply adding their names to the subject list and their data to the proper folders.

```
connectivity = pe.Workflow(name="connectivity")

connectivity.base_dir = op.abspath('dmri_connectivity_advanced')
connectivity.connect([
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (datasource, mapping, [('dwi', 'inputnode.dwi'),
                           ('bvals', 'inputnode.bvals'),
                           ('bvecs', 'inputnode.bvecs')
                           ]),
    (infosource, mapping, [('subject_id', 'inputnode.subject_id')])
])
```

The following functions run the whole workflow and produce a .dot and .png graph of the processing pipeline.

```
if __name__ == '__main__':
    connectivity.run()
    connectivity.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

dmRI: DTI - Diffusion Toolkit, FSL

A pipeline example that uses several interfaces to perform analysis on diffusion weighted images using Diffusion Toolkit tools.

This tutorial is based on the 2010 FSL course and uses data freely available at the FSL website at: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

More details can be found at <http://www.fmrib.ox.ac.uk/fslcourse/lectures/practicals/fdt/index.htm>

In order to run this tutorial you need to have Diffusion Toolkit and FSL tools installed and accessible from matlab/command line. Check by calling `fslinfo` and `dtk` from the command line.

Tell python where to find the appropriate functions.

```
import nipype.interfaces.io as nio          # Data i/o
import nipype.interfaces.fsl as fsl        # fsl
import nipype.interfaces.diffusion_toolkit as dtk
import nipype.interfaces.utility as util    # utility
import nipype.pipeline.engine as pe        # pypeline engine
import os                                   # system functions
from nipype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline
```

Confirm package dependencies are installed. (This is only for the tutorial, rarely would you put this in your own code.)

```
from nipype.utils.misc import package_check

package_check('numpy', '1.3', 'tutorial1')
package_check('scipy', '0.7', 'tutorial1')
package_check('networkx', '1.0', 'tutorial1')
package_check('IPython', '0.10', 'tutorial1')
```

Setting up workflows

This is a generic workflow for DTI data analysis using the FSL

Data specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, `dwis1` and `dwis2`. Each subject directory contains each of the following files: `bvec`, `bval`, diffusion weighted data, a set of target

masks, a seed file, and a transformation matrix.

Below we set some variables to inform the `datasource` about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (dwi or bvals). These fields become the output fields of the `datasource` node in the pipeline.

Specify the subject directories

```
subject_list = ['subj1']
```

Map field names to individual subject runs

```
info = dict(dwi=[['subject_id', 'data']],
            bvecs=[['subject_id', 'bvecs']],
            bvals=[['subject_id', 'bvals']])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                    name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipy.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipy.pipeline.engine.Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                              outfields=list(info.keys())),
                    name='datasource')

datasource.inputs.template = "%s/%s"

# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
datasource.inputs.base_directory = os.path.abspath('fsl_course_data/fdt/')

datasource.inputs.field_template = dict(dwi='%s/%s.nii.gz')
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Setup for Diffusion Tensor Computation

Here we will create a generic workflow for DTI computation

```
computeTensor = pe.Workflow(name='computeTensor')
```

extract the volume with `b=0` (nodif_brain)

```
fslroi = pe.Node(interface=fsl.ExtractROI(), name='fslroi')
fslroi.inputs.t_min = 0
fslroi.inputs.t_size = 1
```

create a brain mask from the `nodif_brain`

```
bet = pe.Node(interface=fsl.BET(), name='bet')
bet.inputs.mask = True
bet.inputs.frac = 0.34
```

correct the diffusion weighted images for eddy_currents

```
eddycorrect = create_eddy_correct_pipeline('eddycorrect')
eddycorrect.inputs.inputnode.ref_num = 0
```

compute the diffusion tensor in each voxel

```
dtifit = pe.Node(interface=dtk.DTIRecon(), name='dtifit')
```

connect all the nodes for this workflow

```
computeTensor.connect([
    (fslroi, bet, [('roi_file', 'in_file')]),
    (eddycorrect, dtifit, [('outputnode.eddy_corrected', 'DWI')])
])
```

Setup for Tractography

Here we will create a workflow to enable deterministic tractography

```
tractography = pe.Workflow(name='tractography')

dtk_tracker = pe.Node(interface=dtk.DTITracker(), name="dtk_tracker")
dtk_tracker.inputs.invert_x = True

smooth_trk = pe.Node(interface=dtk.SplineFilter(), name="smooth_trk")
smooth_trk.inputs.step_length = 0.5
```

connect all the nodes for this workflow

```
tractography.connect([
    (dtk_tracker, smooth_trk, [('track_file', 'track_file')])
])
```

Setup data storage area

```
datasink = pe.Node(interface=nio.DataSink(), name='datasink')
datasink.inputs.base_directory = os.path.abspath('dtiresults')

def getstripdir(subject_id):
    return os.path.join(os.path.abspath('data/workingdir/dwiproc'), '_subject_id_
↪%s' % subject_id)
```

Setup the pipeline that combines the two workflows: tractography and computeTensor

```
dwiproc = pe.Workflow(name="dwiproc")
dwiproc.base_dir = os.path.abspath('dtk_dti_tutorial')
dwiproc.connect([
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (datasource, computeTensor, [('dwi', 'fslroi.in_file'),
                                  ('bvals', 'dtifit.bvals'),
                                  ('bvecs', 'dtifit.bvecs'),
                                  ('dwi', 'eddycorrect.inputnode.in_file')]),
    (computeTensor, tractography, [('bet.mask_file', 'dtk_tracker.mask1_file'),
```

```
                                ('dtifit.tensor', 'dtk_tracker.tensor_file')
                                ])
    ])

    if __name__ == '__main__':
        dwiproc.run()
        dwiproc.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

dmRI: HARDI - Diffusion Toolkit, FSL

A pipeline example that uses several interfaces to perform analysis on diffusion weighted images using Diffusion Toolkit tools.

This tutorial is based on the 2010 FSL course and uses data freely available at the FSL website at: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

More details can be found at <http://www.fmrib.ox.ac.uk/fslcourse/lectures/practicals/fdt/index.htm>

In order to run this tutorial you need to have Diffusion Toolkit and FSL tools installed and accessible from matlab/command line. Check by calling `fsinfo` and `dtk` from the command line.

Tell python where to find the appropriate functions.

```
import nipype.interfaces.io as nio          # Data i/o
import nipype.interfaces.fsl as fsl        # fsl
import nipype.interfaces.diffusion_toolkit as dtk
import nipype.interfaces.utility as util    # utility
import nipype.pipeline.engine as pe        # pypeline engine
import os                                   # system functions
from nipype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline
```

Confirm package dependencies are installed. (This is only for the tutorial, rarely would you put this in your own code.)

```
from nipype.utils.misc import package_check

package_check('numpy', '1.3', 'tutorial1')
package_check('scipy', '0.7', 'tutorial1')
package_check('networkx', '1.0', 'tutorial1')
package_check('IPython', '0.10', 'tutorial1')
```

Setting up workflows

This is a generic workflow for DTI data analysis using the FSL

Data specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, `dwis1` and `dwis2`. Each subject directory contains each of the following files: `bvec`, `bval`, diffusion weighted data, a set of target

masks, a seed file, and a transformation matrix.

Below we set some variables to inform the `datasource` about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (dwi or bvals). These fields become the output fields of the `datasource` node in the pipeline.

Specify the subject directories

```
subject_list = ['siemens_hardi_test']
```

Map field names to individual subject runs

```
info = dict(dwi=[['subject_id', 'siemens_hardi_test_data']],
            bvecs=[['subject_id', 'siemens_hardi_test_data.bvec']],
            bvals=[['subject_id', 'siemens_hardi_test_data.bval']])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                     name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipy.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipy.pipeline.engine.Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                              outfields=list(info.keys())),
                    name='datasource')

datasource.inputs.template = "%s/%s"

# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
datasource.inputs.base_directory = os.path.abspath('data')

datasource.inputs.field_template = dict(dwi='%s/%s.nii')
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Setup for ODF Computation

Here we will create a generic workflow for ODF computation

```
compute_ODF = pe.Workflow(name='compute_ODF')
```

extract the volume with `b=0` (nodif_brain)

```
fslroi = pe.Node(interface=fsl.ExtractROI(), name='fslroi')
fslroi.inputs.t_min = 0
fslroi.inputs.t_size = 1
```

create a brain mask from the `nodif_brain`

```
bet = pe.Node(interface=fsl.BET(), name='bet')
bet.inputs.mask = True
bet.inputs.frac = 0.34
```

correct the diffusion weighted images for eddy_currents

```
eddycorrect = create_eddy_correct_pipeline('eddycorrect')
eddycorrect.inputs.inputnode.ref_num = 0

hardi_mat = pe.Node(interface=dtk.HARDIMat(), name='hardi_mat')

odf_recon = pe.Node(interface=dtk.ODFRecon(), name='odf_recon')
```

connect all the nodes for this workflow

```
compute_ODF.connect([
    (fslroi, bet, [('roi_file', 'in_file')]),
    (eddycorrect, odf_recon, [('outputnode.eddy_corrected', 'DWI')]),
    (eddycorrect, hardi_mat, [('outputnode.eddy_corrected', 'reference_file')]),
    (hardi_mat, odf_recon, [('out_file', 'matrix')])
])
```

Setup for Tracktography

Here we will create a workflow to enable deterministic tracktography

```
tractography = pe.Workflow(name='tractography')

odf_tracker = pe.Node(interface=dtk.ODFTracker(), name="odf_tracker")

smooth_trk = pe.Node(interface=dtk.SplineFilter(), name="smooth_trk")
smooth_trk.inputs.step_length = 1
```

connect all the nodes for this workflow

```
tractography.connect([
    (odf_tracker, smooth_trk, [('track_file', 'track_file')])
])
```

Setup the pipeline that combines the two workflows: tractography and compute_ODF

```
dwiproc = pe.Workflow(name="dwiproc")
dwiproc.base_dir = os.path.abspath('dtk_odf_tutorial')
dwiproc.connect([
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (datasource, compute_ODF, [('dwi', 'fslroi.in_file'),
                                ('bvals', 'hardi_mat.bvals'),
                                ('bvecs', 'hardi_mat.bvecs'),
                                ('dwi', 'eddycorrect.inputnode.in_file')]),
    (compute_ODF, tractography, [('bet.mask_file', 'odf_tracker.mask1_file'),
                                ('odf_recon.ODF', 'odf_tracker.ODF'),
                                ('odf_recon.max', 'odf_tracker.max')])
])

dwiproc.inputs.compute_ODF.hardi_mat.oblique_correction = True
dwiproc.inputs.compute_ODF.odf_recon.n_directions = 31
```

```
dwiproc.inputs.compute_ODF.odf_recon.n_b0 = 5
dwiproc.inputs.compute_ODF.odf_recon.n_output_directions = 181

if __name__ == '__main__':
    dwiproc.run()
    dwiproc.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

CHAPTER 16

dMRI: DTI, FSL

A pipeline example that uses several interfaces to perform analysis on diffusion weighted images using FSL FDT tools.

This tutorial is based on the 2010 FSL course and uses data freely available at the FSL website at: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

More details can be found at <http://www.fmrib.ox.ac.uk/fslcourse/lectures/practicals/fdt/index.htm>

In order to run this tutorial you need to have fsl tools installed and accessible from matlab/command line. Check by calling `fslinfo` from the command line.

Tell python where to find the appropriate functions.

```
import nipype.interfaces.io as nio           # Data i/o
import nipype.interfaces.fsl as fsl          # fsl
import nipype.interfaces.utility as util      # utility
import nipype.pipeline.engine as pe          # pypeline engine
import os                                     # system functions
from nipype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline, \
    create_bedpostx_pipeline
```

Confirm package dependencies are installed. (This is only for the tutorial, rarely would you put this in your own code.)

```
from nipype.utils.misc import package_check

package_check('numpy', '1.3', 'tutorial1')
package_check('scipy', '0.7', 'tutorial1')
package_check('networkx', '1.0', 'tutorial1')
package_check('IPython', '0.10', 'tutorial1')
```

Setting up workflows

This is a generic workflow for DTI data analysis using the FSL

Data specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, `dwis1` and `dwis2`. Each subject directory contains each of the following files: `bvec`, `bval`, diffusion weighted data, a set of target

masks, a seed file, and a transformation matrix.

Below we set some variables to inform the `datasource` about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (dwi or bvals). These fields become the output fields of the `datasource` node in the pipeline.

Specify the subject directories

```
subject_list = ['subj1']
```

Map field names to individual subject runs

```
info = dict(dwi=[['subject_id', 'data']],
            bvecs=[['subject_id', 'bvecs']],
            bvals=[['subject_id', 'bvals']],
            seed_file=[['subject_id', 'MASK_average_thal_right']],
            target_masks=[['subject_id', ['MASK_average_M1_right',
                                         'MASK_average_S1_right',
                                         'MASK_average_occipital_right',
                                         'MASK_average_pfc_right',
                                         'MASK_average_pmc_right',
                                         'MASK_average_ppc_right',
                                         'MASK_average_temporal_right']]])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                     name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipy.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipy.pipeline.engine.Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                              outfields=list(info.keys())),
                    name='datasource')

datasource.inputs.template = "%s/%s"

# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
datasource.inputs.base_directory = os.path.abspath('fsl_course_data/fdt/')

datasource.inputs.field_template = dict(dwi='%s/%s.nii.gz',
                                       seed_file="%s.bedpostX/%s.nii.gz",
                                       target_masks="%s.bedpostX/%s.nii.gz")

datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Setup for Diffusion Tensor Computation

Here we will create a generic workflow for DTI computation

```
computeTensor = pe.Workflow(name='computeTensor')
```

extract the volume with `b=0` (`nodif_brain`)

```
fslroi = pe.Node(interface=fsl.ExtractROI(), name='fslroi')
fslroi.inputs.t_min = 0
fslroi.inputs.t_size = 1
```

create a brain mask from the nodif_brain

```
bet = pe.Node(interface=fsl.BET(), name='bet')
bet.inputs.mask = True
bet.inputs.frac = 0.34
```

correct the diffusion weighted images for eddy_currents

```
eddycorrect = create_eddy_correct_pipeline('eddycorrect')
eddycorrect.inputs.inputnode.ref_num = 0
```

compute the diffusion tensor in each voxel

```
dtifit = pe.Node(interface=fsl.DTIFit(), name='dtifit')
```

connect all the nodes for this workflow

```
computeTensor.connect([
    (fslroi, bet, [('roi_file', 'in_file')]),
    (eddycorrect, dtifit, [('outputnode.eddy_corrected', 'dwi')]),
    (infosource, dtifit, [['subject_id', 'base_name']]),
    (bet, dtifit, [('mask_file', 'mask')])
])
```

Setup for Tracktography

Here we will create a workflow to enable probabilistic tracktography and hard segmentation of the seed region

```
tractography = pe.Workflow(name='tractography')
tractography.base_dir = os.path.abspath('fsl_dti_tutorial')
```

estimate the diffusion parameters: phi, theta, and so on

```
bedpostx = create_bedpostx_pipeline()
bedpostx.get_node("xfibres").iterables = ("n_fibres", [1, 2])

flirt = pe.Node(interface=fsl.FLIRT(), name='flirt')
flirt.inputs.in_file = fsl.Info.standard_image('MNI152_T1_2mm_brain.nii.gz')
flirt.inputs.dof = 12
```

perform probabilistic tracktography

```
probtrackx = pe.Node(interface=fsl.ProbTrackX(), name='probtrackx')
probtrackx.inputs.mode = 'seedmask'
probtrackx.inputs.c_thresh = 0.2
probtrackx.inputs.n_steps = 2000
probtrackx.inputs.step_length = 0.5
probtrackx.inputs.n_samples = 5000
probtrackx.inputs.opd = True
probtrackx.inputs.os2t = True
probtrackx.inputs.loop_check = True
```

perform hard segmentation on the output of probtrackx

```
findthebiggest = pe.Node(interface=fsl.FindTheBiggest(), name='findthebiggest')
```

connect all the nodes for this workflow

```
tractography.add_nodes([bedpostx, flirt])
tractography.connect([(bedpostx, probtrackx, [('outputnode.thsamples', 'thsamples'
↪)],
                                ('outputnode.phsamples', 'phsamples'
↪)],
                                ('outputnode.fsamples', 'fsamples')
                                ]),
                    (probtrackx, findthebiggest, [('targets', 'in_files')]),
                    (flirt, probtrackx, [('out_matrix_file', 'xfm')])
                    ])
```

Setup data storage area

```
datasink = pe.Node(interface=nio.DataSink(), name='datasink')
datasink.inputs.base_directory = os.path.abspath('dtireresults')

def getstripdir(subject_id):
    import os
    return os.path.join(os.path.abspath('data/workingdir/dwiproc'), '_subject_id_
↪%s' % subject_id)
```

Setup the pipeline that combines the two workflows: tractography and computeTensor

```
dwiproc = pe.Workflow(name="dwiproc")
dwiproc.base_dir = os.path.abspath('fsl_dti_tutorial')
dwiproc.connect([
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (datasource, computeTensor, [('dwi', 'fslroi.in_file'),
                                ('bvals', 'dtifit.bvals'),
                                ('bvecs', 'dtifit.bvecs'),
                                ('dwi', 'eddycorrect.inputnode.in_file')]),
    (datasource, tractography, [('bvals', 'bedpostx.inputnode.bvals'),
                                ('bvecs', 'bedpostx.inputnode.bvecs'),
                                ('seed_file', 'probtrackx.seed'),
                                ('target_masks', 'probtrackx.target_masks')
                                ]),
    (computeTensor, tractography, [('eddycorrect.outputnode.eddy_corrected',
↪'bedpostx.inputnode.dwi'),
                                ('bet.mask_file', 'bedpostx.inputnode.mask'),
                                ('bet.mask_file', 'probtrackx.mask'),
                                ('fslroi.roi_file', 'flirt.reference')]),
    (infosource, datasink, [('subject_id', 'container'),
                            (('subject_id', getstripdir), 'strip_dir')]),
    (tractography, datasink, [('findthebiggest.out_file', 'fbiggest.
↪@biggestsegmentation')])
])

if __name__ == '__main__':
    dwiproc.run()
    dwiproc.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipyte source distribution under the `examples` directory.

dmMRI: Group connectivity - Camino, FSL, FreeSurfer

Introduction

This script, `dmri_group_connectivity_camino.py`, runs group-based connectivity analysis using the `dmri.camino.connectivity_mapping` Nipype workflow. Further detail on the processing can be found in *dmMRI: Connectivity - Camino, CMTK, FreeSurfer*. This tutorial can be run using:

```
python dmri_group_connectivity_camino.py
```

We perform this analysis using one healthy subject and two subjects who suffer from Parkinson's disease. The whole package (960 mb as `.tar.gz` / 1.3 gb uncompressed) including the FreeSurfer directories for these subjects, can be acquired from here:

- <http://db.tt/b6F1t0QV>

A data package containing the outputs of this pipeline can be obtained from here:

- <http://db.tt/kNvAI751>

Along with Camino, Camino-Trackvis, FSL, and FreeSurfer, you must also have the Connectome File Format library installed as well as the Connectome Mapper.

- Camino: <http://web4.cs.ucl.ac.uk/research/medic/camino/pmwiki/pmwiki.php?n=Main.HomePage>
- Camino-Trackvis: <http://www.nitrc.org/projects/camino-trackvis/>
- FSL: <http://www.fmrib.ox.ac.uk/fsl/>
- FreeSurfer: <http://surfer.nmr.mgh.harvard.edu/>
- CMTK: <http://www.cmtk.org/>
- CFF: `sudo apt-get install python-cfflib`

Or on github at:

- CFFlib: <https://github.com/LTS5/cfflib>
- CMP: <https://github.com/LTS5/cmp>
- ConnectomeViewer: <https://github.com/LTS5/connectomeviewer>
- TrackVis: <http://trackvis.org/>

The fiber data is available in Numpy arrays, and the connectivity matrix is also produced as a MATLAB matrix.

Import the workflows

First, we import the necessary modules from nipype.

```
import nipyne.interfaces.fsl as fsl
import nipyne.interfaces.freesurfer as fs      # freesurfer
import os.path as op                          # system functions
import cmp
from nipyne.workflows.dmri.camino.group_connectivity import create_group_
↳connectivity_pipeline
from nipyne.workflows.dmri.connectivity.group_connectivity import (create_merge_
↳networks_by_group_workflow,
                                                                    create_merge_
↳group_networks_workflow, create_average_networks_by_group_workflow)
```

Set the proper directories

First, we import the necessary modules from nipyne.

```
fs_dir = op.abspath('/usr/local/freesurfer')
subjects_dir = op.abspath('groupcondatapackage/subjects/')
data_dir = op.abspath('groupcondatapackage/data/')
fs.FSLCommand.set_default_subjects_dir(subjects_dir)
fsl.FSLCommand.set_default_output_type('NIFTI')
```

Define the groups

Here we define the groups for this study. We would like to search for differences between the healthy subject and the two vegetative patients. The group list is defined as a Python dictionary (see <http://docs.python.org/tutorial/datastructures.html>), with group IDs ('controls', 'parkinsons') as keys, and subject/patient names as values. We set the main output directory as 'groupcon'.

```
group_list = {}
group_list['controls'] = ['cont17']
group_list['parkinsons'] = ['pat10', 'pat20']
```

The output directory must be named as well.

```
global output_dir
output_dir = op.abspath('dmri_group_connectivity_camino')
```

Main processing loop

The title for the final grouped-network connectome file is dependent on the group names. The resulting file for this example is 'parkinsons-controls.cff'. The following code implements the format a-b-c-...x.cff for an arbitrary number of groups.

Warning: The 'info' dictionary below is used to define the input files. In this case, the diffusion weighted image contains the string 'dwi'. The same applies to the b-values and b-vector files, and this must be changed to fit your naming scheme.

This line creates the processing workflow given the information input about the groups and subjects.

See also:

- `nipyne/workflows/dmri/mrtrix/group_connectivity.py`
- `nipyne/workflows/dmri/camino/connectivity_mapping.py`
- *dMRI: Connectivity - Camino, CMTK, FreeSurfer*

The purpose of the second-level workflow is simple: It is used to merge each subject's CFF file into one, so that there is a single file containing all of the networks for each group. This can be useful for performing Network

Brain Statistics using the NBS plugin in ConnectomeViewer.

See also:

http://www.connectomeviewer.org/documentation/users/tutorials/tut_nbs.html

```

title = ''
for idx, group_id in enumerate(group_list.keys()):
    title += group_id
    if not idx == len(list(group_list.keys())) - 1:
        title += '-'

    info = dict(dwi=[['subject_id', 'dti']],
                bvecs=[['subject_id', 'bvecs']],
                bvals=[['subject_id', 'bvals']])

    l1pipeline = create_group_connectivity_pipeline(group_list, group_id, data_
↳dir, subjects_dir, output_dir, info)

    # Here we define the parcellation scheme and the number of tracks to produce
    parcellation_scheme = 'NativeFreesurfer'
    cmp_config = cmp.configuration.PipelineConfiguration()
    cmp_config.parcellation_scheme = parcellation_scheme
    l1pipeline.inputs.connectivity.inputnode.resolution_network_file = cmp_config.
↳get_lausanne_parcellation(parcellation_scheme)['freesurferaparc']['node_
↳information_graphml']

    l1pipeline.run()
    l1pipeline.write_graph(format='eps', graph2use='flat')

    # The second-level pipeline is created here
    l2pipeline = create_merge_networks_by_group_workflow(group_list, group_id,
↳data_dir, subjects_dir, output_dir)
    l2pipeline.run()
    l2pipeline.write_graph(format='eps', graph2use='flat')

```

Now that the for loop is complete there are two grouped CFF files each containing the appropriate subjects. It is also convenient to have every subject in a single CFF file, so that is what the third-level pipeline does.

```

l3pipeline = create_merge_group_networks_workflow(group_list, data_dir, subjects_
↳dir, output_dir, title)
l3pipeline.run()
l3pipeline.write_graph(format='eps', graph2use='flat')

```

The fourth and final workflow averages the networks and saves them in another CFF file

```

l4pipeline = create_average_networks_by_group_workflow(group_list, data_dir,
↳subjects_dir, output_dir, title)
l4pipeline.run()
l4pipeline.write_graph(format='eps', graph2use='flat')

```

Example source code

You can download the full source code of this example. This same script is also included in the Nipy source distribution under the examples directory.

dMRI: Group connectivity - MRtrix, FSL, FreeSurfer

Introduction

This script, `dmri_group_connectivity_mrtrix.py`, runs group-based connectivity analysis using the `dmri.mrtrix.connectivity_mapping` Nipype workflow. Further detail on the processing can be found in *dMRI: Connectivity - MRtrix, CMTK, FreeSurfer*. This tutorial can be run using:

```
python dmri_group_connectivity_mrtrix.py
```

We perform this analysis using one healthy subject and two subjects who suffer from Parkinson's disease. The whole package (960 mb as `.tar.gz` / 1.3 gb uncompressed) including the FreeSurfer directories for these subjects, can be acquired from here:

- <http://db.tt/b6F1t0QV>

A data package containing the outputs of this pipeline can be obtained from here:

- <http://db.tt/elmMnIt1>

Along with MRtrix, FSL, and FreeSurfer, you must also have the Connectome File Format library installed as well as the Connectome Mapper (`cmp`).

- MRtrix: <http://www.brain.org.au/software/mrtrix/>
- FSL: <http://www.fmrib.ox.ac.uk/fsl/>
- FreeSurfer: <http://surfer.nmr.mgh.harvard.edu/>
- CMTK: <http://www.cmtk.org/>
- CFF: `sudo apt-get install python-cfflib`

Or on github at:

- CFFlib: <https://github.com/LTS5/cfflib>
- CMP: <https://github.com/LTS5/cmp>

Output data can be visualized in ConnectomeViewer, TrackVis, Gephi, the MRtrix Viewer (`mrview`), and anything that can view Nifti files.

- ConnectomeViewer: <https://github.com/LTS5/connectomeviewer>
- TrackVis: <http://trackvis.org/>
- Gephi: <http://gephi.org/>

The fiber data is available in Numpy arrays, and the connectivity matrix is also produced as a MATLAB matrix.

Import the workflows

First, we import the necessary modules from nipype.

```
import nipy.interfaces.fsl as fsl
import nipy.interfaces.freesurfer as fs    # freesurfer
import os.path as op                      # system functions
import cmp
from nipy.workflows.dmri.mrtrix.group_connectivity import create_group_
↳connectivity_pipeline
from nipy.workflows.dmri.connectivity.group_connectivity import (create_merge_
↳network_results_by_group_workflow, create_merge_group_network_results_workflow,
↳create_average_networks_by_group_workflow)
```

Set the proper directories

First, we import the necessary modules from nipy.

```
subjects_dir = op.abspath('groupcondatapackage/subjects/')
data_dir = op.abspath('groupcondatapackage/data/')
fs.FSLCommand.set_default_subjects_dir(subjects_dir)
fsl.FSLCommand.set_default_output_type('NIFTI')
```

Define the groups

Here we define the groups for this study. We would like to search for differences between the healthy subject and the two vegetative patients. The group list is defined as a Python dictionary (see <http://docs.python.org/tutorial/datastructures.html>), with group IDs ('controls', 'parkinsons') as keys, and subject/patient names as values. We set the main output directory as 'groupcon'.

```
group_list = {}
group_list['controls'] = ['cont17']
group_list['parkinsons'] = ['pat10', 'pat20']
```

The output directory must be named as well.

```
global output_dir
output_dir = op.abspath('dmri_group_connectivity_mrtrix')
```

Main processing loop

The title for the final grouped-network connectome file is dependent on the group names. The resulting file for this example is 'parkinsons-controls.cff'. The following code implements the format a-b-c-...x.cff for an arbitrary number of groups.

Warning: The 'info' dictionary below is used to define the input files. In this case, the diffusion weighted image contains the string 'dti'. The same applies to the b-values and b-vector files, and this must be changed to fit your naming scheme.

The workflow is created given the information input about the groups and subjects.

See also:

- `nipy/workflows/dmri/mrtrix/group_connectivity.py`
- `nipy/workflows/dmri/mrtrix/connectivity_mapping.py`
- *dMRI: Connectivity - MRtrix, CMTK, FreeSurfer*

We set values for absolute threshold used on the fractional anisotropy map. This is done in order to identify single-fiber voxels. In brains with more damage, however, it may be necessary to reduce the threshold, since their brains are have lower average fractional anisotropy values.

We invert the b-vectors in the encoding file, and set the maximum harmonic order of the pre-tractography spherical deconvolution step. This is done to show how to set inputs that will affect both groups.

Next we create and run the second-level pipeline. The purpose of this workflow is simple: It is used to merge each subject's CFF file into one, so that there is a single file containing all of the networks for each group. This can be useful for performing Network Brain Statistics using the NBS plugin in ConnectomeViewer.

See also:

http://www.connectomeviewer.org/documentation/users/tutorials/tut_nbs.html

```
title = ''
for idx, group_id in enumerate(group_list.keys()):
    title += group_id
    if not idx == len(list(group_list.keys())) - 1:
        title += '-'

    info = dict(dwi=[['subject_id', 'dti']],
                bvecs=[['subject_id', 'bvecs']],
                bvals=[['subject_id', 'bvals']])

    llpipeline = create_group_connectivity_pipeline(group_list, group_id, data_
↳dir, subjects_dir, output_dir, info)

    # Here with invert the b-vectors in the Y direction and set the maximum_
↳harmonic order of the
    # spherical deconvolution step
    llpipeline.inputs.connectivity.mapping.fsl2mrtrix.invert_y = True
    llpipeline.inputs.connectivity.mapping.csdeconv.maximum_harmonic_order = 6

    # Here we define the parcellation scheme and the number of tracks to produce
    parcellation_name = 'scale500'
    llpipeline.inputs.connectivity.mapping.Parcellate.parcellation_name = _
↳parcellation_name
    cmp_config = cmp.configuration.PipelineConfiguration()
    cmp_config.parcellation_scheme = "Lausanne2008"
    llpipeline.inputs.connectivity.mapping.inputnode_within.resolution_network_
↳file = cmp_config._get_lausanne_parcellation('Lausanne2008') [parcellation_name] [
↳'node_information_graphml']
    llpipeline.inputs.connectivity.mapping.probCSDstreamtrack.desired_number_of_
↳tracks = 100000

    llpipeline.run()
    llpipeline.write_graph(format='eps', graph2use='flat')

    # The second-level pipeline is created here
    l2pipeline = create_merge_network_results_by_group_workflow(group_list, group_
↳id, data_dir, subjects_dir, output_dir)
    l2pipeline.inputs.l2inputnode.network_file = cmp_config._get_lausanne_
↳parcellation('Lausanne2008') [parcellation_name] ['node_information_graphml']
    l2pipeline.run()
    l2pipeline.write_graph(format='eps', graph2use='flat')
```

Now that the for loop is complete there are two grouped CFF files each containing the appropriate subjects. It is also convenient to have every subject in a single CFF file, so that is what the third-level pipeline does.

```
l3pipeline = create_merge_group_network_results_workflow(group_list, data_dir, _
↳subjects_dir, output_dir, title)
l3pipeline.run()
l3pipeline.write_graph(format='eps', graph2use='flat')
```

The fourth and final workflow averages the networks and saves them in another CFF file

```
l4pipeline = create_average_networks_by_group_workflow(group_list, data_dir, ↵  
↵subjects_dir, output_dir, title)  
l4pipeline.run()  
l4pipeline.write_graph(format='eps', graph2use='flat')
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

Introduction

This script, `dmri_mrtrix_dti.py`, demonstrates the ability to perform advanced diffusion analysis in a Nipype pipeline:

```
python dmri_mrtrix_dti.py
```

We perform this analysis using the FSL course data, which can be acquired from here:

- http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

Import necessary modules from nipype.

```
import nipype.interfaces.io as nio          # Data i/o
import nipype.interfaces.utility as util    # utility
import nipype.pipeline.engine as pe        # pypeline engine
import nipype.interfaces.mrtrix as mrtrix  # <---- The important new part!
import nipype.interfaces.fsl as fsl
import nipype.algorithms.misc as misc
import os
import os.path as op                       # system functions

fsl.FSLCommand.set_default_output_type('NIFTI')
```

This needs to point to the `fdt` folder you can find after extracting

- http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

```
data_dir = op.abspath(op.join(op.curdir, 'exdata/'))
subject_list = ['subj1']
```

Use `infosource` node to loop through the subject list and define the input files. For our purposes, these are the diffusion-weighted MR image, b vectors, and b values.

```
infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
    ↪name="infosource")
infosource.iterables = ('subject_id', subject_list)

info = dict(dwi=[['subject_id', 'data']],
            bvecs=[['subject_id', 'bvecs']],
            bvals=[['subject_id', 'bvals']])
```

Use datasource node to perform the actual data grabbing. Templates for the associated images are used to obtain the correct images.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                              outfields=list(info.keys())),
                    name='datasource')

datasource.inputs.template = "%s/%s"
datasource.inputs.base_directory = data_dir
datasource.inputs.field_template = dict(dwi='%s/%s.nii.gz')
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

An inputnode is used to pass the data obtained by the data grabber to the actual processing functions

```
inputnode = pe.Node(interface=util.IdentityInterface(fields=["dwi", "bvecs",
↪ "bvals"]), name="inputnode")
```

Diffusion processing nodes

See also:

dmri_connectivity_advanced.py Tutorial with further detail on using MRtrix tractography for connectivity analysis

<http://www.brain.org.au/software/mrtrix/index.html> MRtrix's online documentation

b-values and b-vectors stored in FSL's format are converted into a single encoding file for MRTrix.

```
fsl2mrtrix = pe.Node(interface=mrtrix.FSL2MRtrix(), name='fsl2mrtrix')
```

Tensors are fitted to each voxel in the diffusion-weighted image and from these three maps are created:

Major eigenvector in each voxel

Apparent diffusion coefficient

Fractional anisotropy

```
gunzip = pe.Node(interface=misc.Gunzip(), name='gunzip')
dwi2tensor = pe.Node(interface=mrtrix.DWI2Tensor(), name='dwi2tensor')
tensor2vector = pe.Node(interface=mrtrix.Tensor2Vector(), name='tensor2vector')
tensor2adc = pe.Node(interface=mrtrix.Tensor2ApparentDiffusion(), name='tensor2adc
↪')
tensor2fa = pe.Node(interface=mrtrix.Tensor2FractionalAnisotropy(), name=
↪ 'tensor2fa')
```

These nodes are used to create a rough brain mask from the b0 image. The b0 image is extracted from the original diffusion-weighted image, put through a simple thresholding routine, and smoothed using a 3x3 median filter.

```
MRconvert = pe.Node(interface=mrtrix.MRConvert(), name='MRconvert')
MRconvert.inputs.extract_at_axis = 3
MRconvert.inputs.extract_at_coordinate = [0]
threshold_b0 = pe.Node(interface=mrtrix.Threshold(), name='threshold_b0')
median3d = pe.Node(interface=mrtrix.MedianFilter3D(), name='median3d')
```

The brain mask is also used to help identify single-fiber voxels. This is done by passing the brain mask through two erosion steps, multiplying the remaining mask with the fractional anisotropy map, and thresholding the result to obtain some highly anisotropic within-brain voxels.

```
erode_mask_firstpass = pe.Node(interface=mrtrix.Erode(), name='erode_mask_
↪ firstpass')
erode_mask_secondpass = pe.Node(interface=mrtrix.Erode(), name='erode_mask_
↪ secondpass')
```

```
MRmultiply = pe.Node(interface=mrtrix.MRMultiply(), name='MRmultiply')
MRmult_merge = pe.Node(interface=util.Merge(2), name='MRmultiply_merge')
threshold_FA = pe.Node(interface=mrtrix.Threshold(), name='threshold_FA')
threshold_FA.inputs.absolute_threshold_value = 0.7
```

For whole-brain tracking we also require a broad white-matter seed mask. This is created by generating a white matter mask, given a brainmask, and thresholding it at a reasonably high level.

```
bet = pe.Node(interface=fsl.BET(mask=True), name='bet_b0')
gen_WM_mask = pe.Node(interface=mrtrix.GenerateWhiteMatterMask(), name='gen_WM_
↳mask')
threshold_wmmask = pe.Node(interface=mrtrix.Threshold(), name='threshold_wmmask')
threshold_wmmask.inputs.absolute_threshold_value = 0.4
```

The spherical deconvolution step depends on the estimate of the response function in the highly anisotropic voxels we obtained above.

Warning: For damaged or pathological brains one should take care to lower the maximum harmonic order of these steps.

```
estimatereponse = pe.Node(interface=mrtrix.EstimateResponseForSH(), name=
↳'estimatereponse')
estimatereponse.inputs.maximum_harmonic_order = 6
csdeconv = pe.Node(interface=mrtrix.ConstrainedSphericalDeconvolution(), name=
↳'csdeconv')
csdeconv.inputs.maximum_harmonic_order = 6
```

Finally, we track probabilistically using the orientation distribution functions obtained earlier. The tracts are then used to generate a tract-density image, and they are also converted to TrackVis format.

```
probCSDstreamtrack = pe.Node(interface=mrtrix.
↳ProbabilisticSphericallyDeconvolutedStreamlineTrack(), name='probCSDstreamtrack
↳')
probCSDstreamtrack.inputs.inputmodel = 'SD_PROB'
probCSDstreamtrack.inputs.maximum_number_of_tracks = 150000
tracks2prob = pe.Node(interface=mrtrix.Tracks2Prob(), name='tracks2prob')
tracks2prob.inputs.colour = True
tck2trk = pe.Node(interface=mrtrix.MRTrix2TrackVis(), name='tck2trk')
```

Creating the workflow

In this section we connect the nodes for the diffusion processing.

```
tractography = pe.Workflow(name='tractography')

tractography.connect([(inputnode, fsl2mrtrix, [("bvecs", "bvec_file"),
("bvals", "bval_file")])])
tractography.connect([(inputnode, gunzip, [("dwi", "in_file")])])
tractography.connect([(gunzip, dwi2tensor, [("out_file", "in_file")])])
tractography.connect([(fsl2mrtrix, dwi2tensor, [("encoding_file", "encoding_file
↳")])])

tractography.connect([(dwi2tensor, tensor2vector, [(['tensor', 'in_file']]),
(dwi2tensor, tensor2adc, [(['tensor', 'in_file']]),
(dwi2tensor, tensor2fa, [(['tensor', 'in_file']]),
])
tractography.connect([(tensor2fa, MRmult_merge, [("FA", "in1")])])
```

This block creates the rough brain mask to be multiplied, multiplies it with the fractional anisotropy image, and thresholds it to get the single-fiber voxels.

```
tractography.connect([(gunzip, MRconvert, [("out_file", "in_file")])])
tractography.connect([(MRconvert, threshold_b0, [("converted", "in_file")])])
tractography.connect([(threshold_b0, median3d, [("out_file", "in_file")])])
tractography.connect([(median3d, erode_mask_firstpass, [("out_file", "in_file")])])
tractography.connect([(erode_mask_firstpass, erode_mask_secondpass, [("out_file", "in_file")])])
tractography.connect([(erode_mask_secondpass, MRmult_merge, [("out_file", "in2")])])
tractography.connect([(MRmult_merge, MRmultiply, [("out", "in_files")])])
tractography.connect([(MRmultiply, threshold_FA, [("out_file", "in_file")])])
```

Here the thresholded white matter mask is created for seeding the tractography.

```
tractography.connect([(gunzip, bet, [("out_file", "in_file")])])
tractography.connect([(gunzip, gen_WM_mask, [("out_file", "in_file")])])
tractography.connect([(bet, gen_WM_mask, [("mask_file", "binary_mask")])])
tractography.connect([(fsl2mrtrix, gen_WM_mask, [("encoding_file", "encoding_file")])])
tractography.connect([(gen_WM_mask, threshold_wmmask, [("WMprobabilitymap", "in_file")])])
```

Next we estimate the fiber response distribution.

```
tractography.connect([(gunzip, estimatorresponse, [("out_file", "in_file")])])
tractography.connect([(fsl2mrtrix, estimatorresponse, [("encoding_file", "encoding_file")])])
tractography.connect([(threshold_FA, estimatorresponse, [("out_file", "mask_image")])])
```

Run constrained spherical deconvolution.

```
tractography.connect([(gunzip, csdeconv, [("out_file", "in_file")])])
tractography.connect([(gen_WM_mask, csdeconv, [("WMprobabilitymap", "mask_image")])])
tractography.connect([(estimatorresponse, csdeconv, [("response", "response_file")])])
tractography.connect([(fsl2mrtrix, csdeconv, [("encoding_file", "encoding_file")])])
```

Connect the tractography and compute the tract density image.

```
tractography.connect([(threshold_wmmask, probCSDstreamtrack, [("out_file", "seed_file")])])
tractography.connect([(csdeconv, probCSDstreamtrack, [("spherical_harmonics_image", "in_file")])])
tractography.connect([(probCSDstreamtrack, tracks2prob, [("tracked", "in_file")])])
tractography.connect([(gunzip, tracks2prob, [("out_file", "template_file")])])

tractography.connect([(gunzip, tck2trk, [("out_file", "image_file")])])
tractography.connect([(probCSDstreamtrack, tck2trk, [("tracked", "in_file")])])
```

Finally, we create another higher-level workflow to connect our tractography workflow with the info and data-grabbing nodes declared at the beginning. Our tutorial is now extensible to any arbitrary number of subjects by simply adding their names to the subject list and their data to the proper folders.

```
dwiproc = pe.Workflow(name="dwiproc")
dwiproc.base_dir = os.path.abspath('dmri_mrtrix_dti')
dwiproc.connect([
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (datasource, tractography, [('dwi', 'inputnode.dwi'),
                                ('bvals', 'inputnode.bvals'),
                                ('bvecs', 'inputnode.bvecs')
                               ])
])

if __name__ == '__main__':
    dwiproc.run()
    dwiproc.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipyype source distribution under the `examples` directory.

dMRI: Preprocessing

Introduction

This script, `dmri_preprocessing.py`, demonstrates how to prepare dMRI data for tractography and connectivity analysis with `nipype`.

We perform this analysis using the FSL course data, which can be acquired from here: http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz

Can be executed in command line using `python dmri_preprocessing.py`

Import necessary modules from `nipype`.

```
import os                                # system functions
import nipype.interfaces.io as nio       # Data i/o
import nipype.interfaces.utility as niu  # utility
import nipype.algorithms.misc as misc

import nipype.pipeline.engine as pe      # pypeline engine

from nipype.interfaces import fsl
from nipype.interfaces import ants
```

Load specific `nipype`'s workflows for preprocessing of dMRI data: `nipype.workflows.dmri.preprocess.epi.all_peb_pipeline`, as data include a *b0* volume with reverse encoding direction (*P*>>>*A*, or *y*), in contrast with the general acquisition encoding that is *A*>>>*P* or *-y* (in RAS systems).

```
from nipype.workflows.dmri.fsl.artifacts import all_fsl_pipeline, remove_bias
```

Map field names into individual subject runs

```
info = dict(dwi=[['subject_id', 'dwi_data']],
            bvecs=[['subject_id', 'bvecs']],
            bvals=[['subject_id', 'bvals']],
            dwi_rev=[['subject_id', 'nodif_PA']])

infosource = pe.Node(interface=niu.IdentityInterface(fields=['subject_id']),
                      name="infosource")

# Set the subject 1 identifier in subject_list,
# we choose the preproc dataset as it contains uncorrected files.
```

```
subject_list = ['subj1_preproc']
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipy.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(nio.DataGrabber(infields=['subject_id'],
                                     outfields=list(info.keys())), name=
    ↪ 'datasource')

datasource.inputs.template = "%s/%s"

# This needs to point to the fdt folder you can find after extracting
# http://www.fmrib.ox.ac.uk/fslcourse/fsl_course_data2.tar.gz
datasource.inputs.base_directory = os.path.abspath('fdt1')
datasource.inputs.field_template = dict(dwi='%s/%s.nii.gz',
                                       dwi_rev='%s/%s.nii.gz')

datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

An `inputnode` is used to pass the data obtained by the data grabber to the actual processing functions

```
inputnode = pe.Node(niu.IdentityInterface(fields=["dwi", "bvecs", "bvals",
                                                  "dwi_rev"]), name="inputnode")
```

Setup for dMRI preprocessing

In this section we initialize the appropriate workflow for preprocessing of diffusion images.

Artifacts correction

We will use the combination of `topup` and `eddy` as suggested by FSL.

In order to configure the susceptibility distortion correction (SDC), we first write the specific parameters of our echo-planar imaging (EPI) images.

Particularly, we look into the `acqparams.txt` file of the selected subject to gather the encoding direction, acceleration factor (in parallel sequences it is > 1), and readout time or echospacing.

```
epi_AP = {'echospacing': 66.5e-3, 'enc_dir': 'y-'}
epi_PA = {'echospacing': 66.5e-3, 'enc_dir': 'y'}
prep = all_fsl_pipeline(epi_params=epi_AP, altepi_params=epi_PA)
```

Bias field correction

Finally, we set up a node to correct for a single multiplicative bias field from computed on the *b0* image, as suggested in [Jeurissen2014].

```
bias = remove_bias()
```

Connect nodes in workflow

We create a higher level workflow to connect the nodes. Please excuse the author for writing the arguments of the connect function in a not-standard style with readability aims.

```
wf = pe.Workflow(name="dMRI_Preprocessing")
wf.base_dir = os.path.abspath('preprocessing_dmri_tutorial')
wf.connect([
    (infosource, datasource, [('subject_id', 'subject_id')]),
    (datasource, prep, [('dwi', 'inputnode.in_file'),
                        ('dwi_rev', 'inputnode.alt_file'),
                        ('bvals', 'inputnode.in_bval'),
                        ('bvecs', 'inputnode.in_bvec')]),
    (prep, bias, [('outputnode.out_file', 'inputnode.in_file'),
                  ('outputnode.out_mask', 'inputnode.in_mask')]),
    (datasource, bias, [('bvals', 'inputnode.in_bval')])
])
```

Run the workflow as command line executable

```
if __name__ == '__main__':
    wf.run()
    wf.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.

dMRI: TBSS on NKI RS data

A pipeline to do a TBSS analysis on the NKI rockland sample data

```
from nipype.workflows.dmri.fsl.dti import create_eddy_correct_pipeline
from nipype.workflows.dmri.fsl.tbss import create_tbss_non_FA, create_tbss_all
```

Tell python where to find the appropriate functions.

```
import nipype.interfaces.io as nio          # Data i/o
import nipype.interfaces.fsl as fsl        # fsl
import nipype.interfaces.utility as util    # utility
import nipype.pipeline.engine as pe        # pypeline engine
import os                                   # system functions

fsl.FSLCommand.set_default_output_type('NIFTI')
```

You can get the data from:

http://fcon_1000.projects.nitrc.org/indi/pro/eNKI_RS_TRT/FrontPage.html

```
dataDir = os.path.abspath('nki_rs_data')
workingdir = './tbss_example'
subjects_list = ['2475376', '3313349', '3808535', '3893245', '8735778',
                 '9630905']

gen_fa = pe.Workflow(name="gen_fa")
gen_fa.base_dir = os.path.join(os.path.abspath(workingdir), '11')

subject_id_infosource = pe.Node(util.IdentityInterface(fields=['subject_id']),
                                name='subject_id_infosource')
subject_id_infosource.iterables = ('subject_id', subjects_list)

datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                                outfields=['dwi', 'bvec',
                                                            'bval']),
                      name='datasource')
datasource.inputs.base_directory = os.path.abspath(dataDir)
datasource.inputs.template = '%s/session2/DTI_mx_137/dti.%s'
datasource.inputs.template_args = dict(dwi=[['subject_id', 'nii.gz']],
                                       bvec=[['subject_id', 'bvec']],
                                       bval=[['subject_id', 'bval']])
```

```

datasource.inputs.sort_filelist = True
gen_fa.connect(subject_id_infosource, 'subject_id', datasource, 'subject_id')

eddy_correct = create_eddy_correct_pipeline()
eddy_correct.inputs.inputnode.ref_num = 0
gen_fa.connect(datasource, 'dwi', eddy_correct, 'inputnode.in_file')

bet = pe.Node(interface=fsl.BET(), name='bet')
bet.inputs.mask = True
bet.inputs.frac = 0.34
gen_fa.connect(eddy_correct, 'pick_ref.out', bet, 'in_file')

dtifit = pe.Node(interface=fsl.DTIFit(), name='dtifit')
gen_fa.connect(eddy_correct, 'outputnode.eddy_corrected', dtifit, 'dwi')
gen_fa.connect(subject_id_infosource, 'subject_id', dtifit, 'base_name')
gen_fa.connect(bet, 'mask_file', dtifit, 'mask')
gen_fa.connect(datasource, 'bvec', dtifit, 'bvecs')
gen_fa.connect(datasource, 'bval', dtifit, 'bvals')

datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.join(os.path.abspath(workingdir),
                                              'l1_results')

datasink.inputs.parameterization = False
gen_fa.connect(dtifit, 'FA', datasink, 'FA')
gen_fa.connect(dtifit, 'MD', datasink, 'MD')

if __name__ == '__main__':
    gen_fa.write_graph()
    gen_fa.run()

```

Here we get the FA list including all the subjects.

```

tbss_source = pe.Node(interface=nio.DataGrabber(outfiles=['fa_list',
                                                         'md_list']),
                      name='tbss_source')
tbss_source.inputs.base_directory = datasink.inputs.base_directory
tbss_source.inputs.template = '%s/%s_%s.nii'
tbss_source.inputs.template_args = dict(fa_list=[['FA', subjects_list, 'FA']],
                                         md_list=[['MD', subjects_list, 'MD']])
tbss_source.inputs.sort_filelist = True

```

TBSS analysis

```

tbss_all = create_tbss_all()
tbss_all.inputs.inputnode.skeleton_thresh = 0.2

tbssproc = pe.Workflow(name="tbssproc")
tbssproc.base_dir = os.path.join(os.path.abspath(workingdir), 'l2')
tbssproc.connect(tbss_source, 'fa_list', tbss_all, 'inputnode.fa_list')

tbss_MD = create_tbss_non_FA(name='tbss_MD')
tbss_MD.inputs.inputnode.skeleton_thresh = tbss_all.inputs.inputnode.skeleton_
↳ thresh

tbssproc.connect([(tbss_all, tbss_MD, [
    ('tbss2.outputnode.field_list',
     'inputnode.field_list'),
    ('tbss3.outputnode.groupmask',
     'inputnode.groupmask'),
    ('tbss3.outputnode.meanfa_file',

```

```
        'inputnode.meanfa_file'),
        ('tbss4.outputnode.distance_map',
         'inputnode.distance_map']],
        (tbss_source, tbss_MD, [('md_list',
                                'inputnode.file_list'))],
        ])

if __name__ == '__main__':
    tbssproc.write_graph()
    tbssproc.run()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

fmRI: OpenfMRI.org data, FSL, ANTS, c3daffine

A growing number of datasets are available on [OpenfMRI](#). This script demonstrates how to use nipy to analyze a data set:

```
python fmri_ants_openfmri.py --datasetdir ds107
```

```
from __future__ import division, unicode_literals
from builtins import open, range, str, bytes

from glob import glob
import os

from nipy import config
from nipy import LooseVersion
from nipy import Workflow, Node, MapNode
from nipy.utils.filemanip import filename_to_list
import nipy.pipeline.engine as pe
import nipy.algorithms.modelgen as model
import nipy.algorithms.rapidart as ra
from nipy.algorithms.misc import TSNR
from nipy.interfaces.c3 import C3dAffineTool
from nipy.interfaces import fsl, Function, ants, freesurfer as fs
import nipy.interfaces.io as nio
from nipy.interfaces.io import FreeSurferSource
import nipy.interfaces.utility as niu
from nipy.interfaces.utility import Merge, IdentityInterface
from nipy.workflows.fmri.fsl import (create_featreg_preproc,
                                     create_modelfit_workflow,
                                     create_fixed_effects_flow)

from nipy.utils import NUMPY_MMAP

config.enable_provenance()
version = 0
if (fsl.Info.version() and LooseVersion(fsl.Info.version()) > LooseVersion('5.0.6
↪')):
    version = 507
```

```

fsl.FSLCommand.set_default_output_type('NIFTI_GZ')

imports = [
    'import os',
    'import nibabel as nb',
    'import numpy as np',
    'import scipy as sp',
    'from nipy.utils.filemanip import filename_to_list, list_to_filename, split_
    ↪filename',
    'from scipy.special import legendre'
]

def median(in_files):
    """Computes an average of the median of each realigned timeseries

    Parameters
    -----

    in_files: one or more realigned Nifti 4D time series

    Returns
    -----

    out_file: a 3D Nifti file
    """
    average = None
    for idx, filename in enumerate(filename_to_list(in_files)):
        img = nb.load(filename, mmap=NUMPY_MMAP)
        data = np.median(img.get_data(), axis=3)
        if average is None:
            average = data
        else:
            average = average + data
    median_img = nb.Nifti1Image(average / float(idx + 1), img.affine,
                                img.header)
    filename = os.path.join(os.getcwd(), 'median.nii.gz')
    median_img.to_filename(filename)
    return filename

def create_reg_workflow(name='registration'):
    """Create a FEAT preprocessing workflow together with freesurfer

    Parameters
    -----

    name : name of workflow (default: 'registration')

    Inputs:

        inputspec.source_files : files (filename or list of filenames to register)
        inputspec.mean_image : reference image to use
        inputspec.anatomical_image : anatomical image to coregister to
        inputspec.target_image : registration target

    Outputs:

        outputspec.func2anat_transform : FLIRT transform
        outputspec.anat2target_transform : FLIRT+FNIRT transform

```

```

        outputspec.transformed_files : transformed files in target space
        outputspec.transformed_mean : mean image in target space
    """

    register = pe.Workflow(name=name)

    inputnode = pe.Node(interface=niu.IdentityInterface(fields=['source_files',
                                                                'mean_image',
                                                                'anatomical_image',
                                                                'target_image',
                                                                'target_image_',
                                                                'config_file'])),
        name='inputspec')

    outputnode = pe.Node(interface=niu.IdentityInterface(fields=['func2anat_',
                                                                'anat2target_',
                                                                'transformed_',
                                                                'transformed_mean',
                                                                'anat2target',
                                                                'mean2anat_mask'
                                                                ])),
        name='outputspec')

```

Estimate the tissue classes from the anatomical image. But use spm's segment as FSL appears to be breaking.

```

stripper = pe.Node(fsl.BET(), name='stripper')
register.connect(inputnode, 'anatomical_image', stripper, 'in_file')
fast = pe.Node(fsl.FAST(), name='fast')
register.connect(stripper, 'out_file', fast, 'in_files')

```

Binarize the segmentation

```

binarize = pe.Node(fsl.ImageMaths(op_string='-nan -thr 0.5 -bin'),
    name='binarize')
pickindex = lambda x, i: x[i]
register.connect(fast, ('partial_volume_files', pickindex, 2),
    binarize, 'in_file')

```

Calculate rigid transform from mean image to anatomical image

```

mean2anat = pe.Node(fsl.FLIRT(), name='mean2anat')
mean2anat.inputs.dof = 6
register.connect(inputnode, 'mean_image', mean2anat, 'in_file')
register.connect(stripper, 'out_file', mean2anat, 'reference')

```

Now use bbr cost function to improve the transform

```

mean2anatbbr = pe.Node(fsl.FLIRT(), name='mean2anatbbr')
mean2anatbbr.inputs.dof = 6
mean2anatbbr.inputs.cost = 'bbr'
mean2anatbbr.inputs.schedule = os.path.join(os.getenv('FSLDIR'),
    'etc/flirtsch/bbr.sch')
register.connect(inputnode, 'mean_image', mean2anatbbr, 'in_file')
register.connect(binimize, 'out_file', mean2anatbbr, 'wm_seg')
register.connect(inputnode, 'anatomical_image', mean2anatbbr, 'reference')

```

```
register.connect(mean2anat, 'out_matrix_file',
                 mean2anatbbr, 'in_matrix_file')
```

Create a mask of the median image coregistered to the anatomical image

```
mean2anat_mask = Node(fsl.BET(mask=True), name='mean2anat_mask')
register.connect(mean2anatbbr, 'out_file', mean2anat_mask, 'in_file')
```

Convert the BBRegister transformation to ANTS ITK format

```
convert2itk = pe.Node(C3dAffineTool(),
                      name='convert2itk')
convert2itk.inputs.fsl2ras = True
convert2itk.inputs.itk_transform = True
register.connect(mean2anatbbr, 'out_matrix_file', convert2itk, 'transform_file')
register.connect(inputnode, 'mean_image', convert2itk, 'source_file')
register.connect(stripper, 'out_file', convert2itk, 'reference_file')
```

Compute registration between the subject's structural and MNI template This is currently set to perform a very quick registration. However, the registration can be made significantly more accurate for cortical structures by increasing the number of iterations All parameters are set using the example from: <https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh>

```
reg = pe.Node(ants.Registration(), name='antsRegister')
reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = ['Rigid', 'Affine', 'SyN']
reg.inputs.transform_parameters = [(0.1,), (0.1,), (0.2, 3.0, 0.0)]
reg.inputs.number_of_iterations = [[10000, 11110, 11110]] * 2 + [[100, 30, 20]]
reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = True
reg.inputs.initial_moving_transform_com = True
reg.inputs.metric = ['Mattes'] * 2 + [['Mattes', 'CC']]
reg.inputs.metric_weight = [1] * 2 + [[0.5, 0.5]]
reg.inputs.radius_or_number_of_bins = [32] * 2 + [[32, 4]]
reg.inputs.sampling_strategy = ['Regular'] * 2 + [[None, None]]
reg.inputs.sampling_percentage = [0.3] * 2 + [[None, None]]
reg.inputs.convergence_threshold = [1.e-8] * 2 + [-0.01]
reg.inputs.convergence_window_size = [20] * 2 + [5]
reg.inputs.smoothing_sigmas = [[4, 2, 1]] * 2 + [[1, 0.5, 0]]
reg.inputs.sigma_units = ['vox'] * 3
reg.inputs.shrink_factors = [[3, 2, 1]] * 2 + [[4, 2, 1]]
reg.inputs.use_estimate_learning_rate_once = [True] * 3
reg.inputs.use_histogram_matching = [False] * 2 + [True]
reg.inputs.winsorize_lower_quantile = 0.005
reg.inputs.winsorize_upper_quantile = 0.995
reg.inputs.args = '--float'
reg.inputs.output_warped_image = 'output_warped_image.nii.gz'
reg.inputs.num_threads = 4
reg.plugin_args = {'qsub_args': '-pe orte 4',
                  'sbatch_args': '--mem=6G -c 4'}
register.connect(stripper, 'out_file', reg, 'moving_image')
register.connect(inputnode, 'target_image_brain', reg, 'fixed_image')
```

Concatenate the affine and ants transforms into a list

```
pickfirst = lambda x: x[0]

merge = pe.Node(niu.Merge(2), iterfield=['in2'], name='mergexfm')
register.connect(convert2itk, 'itk_transform', merge, 'in2')
```

```
register.connect(reg, 'composite_transform', merge, 'in1')
```

Transform the mean image. First to anatomical and then to target

```
warpmean = pe.Node(ants.ApplyTransforms(),
                    name='warpmean')
warpmean.inputs.input_image_type = 0
warpmean.inputs.interpolation = 'Linear'
warpmean.inputs.invert_transform_flags = [False, False]
warpmean.inputs.terminal_output = 'file'

register.connect(inputnode, 'target_image_brain', warpmean, 'reference_image')
register.connect(inputnode, 'mean_image', warpmean, 'input_image')
register.connect(merge, 'out', warpmean, 'transforms')
```

Transform the remaining images. First to anatomical and then to target

```
warpall = pe.MapNode(ants.ApplyTransforms(),
                     iterfield=['input_image'],
                     name='warpall')
warpall.inputs.input_image_type = 0
warpall.inputs.interpolation = 'Linear'
warpall.inputs.invert_transform_flags = [False, False]
warpall.inputs.terminal_output = 'file'

register.connect(inputnode, 'target_image_brain', warpall, 'reference_image')
register.connect(inputnode, 'source_files', warpall, 'input_image')
register.connect(merge, 'out', warpall, 'transforms')
```

Assign all the output files

```
register.connect(reg, 'warped_image', outputnode, 'anat2target')
register.connect(warpmean, 'output_image', outputnode, 'transformed_mean')
register.connect(warpall, 'output_image', outputnode, 'transformed_files')
register.connect(mean2anatbbr, 'out_matrix_file',
                 outputnode, 'func2anat_transform')
register.connect(mean2anat_mask, 'mask_file',
                 outputnode, 'mean2anat_mask')
register.connect(reg, 'composite_transform',
                 outputnode, 'anat2target_transform')

return register

def get_aparc_aseg(files):
    """Return the aparc+aseg.mgz file"""
    for name in files:
        if 'aparc+aseg.mgz' in name:
            return name
    raise ValueError('aparc+aseg.mgz not found')

def create_fs_reg_workflow(name='registration'):
    """Create a FEAT preprocessing workflow together with freesurfer

    Parameters
    -----

    name : name of workflow (default: 'registration')
```

```

Inputs::

    inputspec.source_files : files (filename or list of filenames to register)
    inputspec.mean_image : reference image to use
    inputspec.target_image : registration target

Outputs::

    outputspec.func2anat_transform : FLIRT transform
    outputspec.anat2target_transform : FLIRT+FNIRT transform
    outputspec.transformed_files : transformed files in target space
    outputspec.transformed_mean : mean image in target space
    """

register = Workflow(name=name)

inputnode = Node(interface=IdentityInterface(fields=['source_files',
                                                    'mean_image',
                                                    'subject_id',
                                                    'subjects_dir',
                                                    'target_image']),
                  name='inputspec')

outputnode = Node(interface=IdentityInterface(fields=['func2anat_transform',
                                                    'out_reg_file',
                                                    'anat2target_transform',
                                                    'transforms',
                                                    'transformed_mean',
                                                    'transformed_files',
                                                    'min_cost_file',
                                                    'anat2target',
                                                    'aparc',
                                                    'mean2anat_mask'
                                                    ]),
                  name='outputspec')

# Get the subject's freesurfer source directory
fssource = Node(FreeSurferSource(),
                name='fssource')
fssource.run_without_submitting = True
register.connect(inputnode, 'subject_id', fssource, 'subject_id')
register.connect(inputnode, 'subjects_dir', fssource, 'subjects_dir')

convert = Node(freesurfer.MRISConvert(out_type='nii'),
               name="convert")
register.connect(fssource, 'T1', convert, 'in_file')

# Coregister the median to the surface
bbregister = Node(freesurfer.BBRegister(registered_file=True),
                  name='bbregister')
bbregister.inputs.init = 'fsl'
bbregister.inputs.contrast_type = 't2'
bbregister.inputs.out_fsl_file = True
bbregister.inputs.epi_mask = True
register.connect(inputnode, 'subject_id', bbregister, 'subject_id')
register.connect(inputnode, 'mean_image', bbregister, 'source_file')
register.connect(inputnode, 'subjects_dir', bbregister, 'subjects_dir')

```

```
# Create a mask of the median coregistered to the anatomical image
mean2anat_mask = Node(fsl.BET(mask=True), name='mean2anat_mask')
register.connect(bregister, 'registered_file', mean2anat_mask, 'in_file')
```

use aparc+aseg's brain mask

```
binarize = Node(fs.Binarize(min=0.5, out_type="nii.gz", dilate=1), name="binarize_
↳aparc")
register.connect(fssource, ("aparc_aseg", get_aparc_aseg), binarize, "in_file")

stripper = Node(fsl.ApplyMask(), name='stripper')
register.connect(binarize, "binary_file", stripper, "mask_file")
register.connect(convert, 'out_file', stripper, 'in_file')
```

Apply inverse transform to aparc file

```
aparcxfm = Node(freesurfer.ApplyVolTransform(inverse=True,
                                             interp='nearest'),
               name='aparc_inverse_transform')
register.connect(inputnode, 'subjects_dir', aparcxfm, 'subjects_dir')
register.connect(bregister, 'out_reg_file', aparcxfm, 'reg_file')
register.connect(fssource, ('aparc_aseg', get_aparc_aseg),
               aparcxfm, 'target_file')
register.connect(inputnode, 'mean_image', aparcxfm, 'source_file')
```

Convert the BBRegister transformation to ANTS ITK format

```
convert2itk = Node(C3dAffineTool(), name='convert2itk')
convert2itk.inputs.fsl2ras = True
convert2itk.inputs.itk_transform = True
register.connect(bregister, 'out_fsl_file', convert2itk, 'transform_file')
register.connect(inputnode, 'mean_image', convert2itk, 'source_file')
register.connect(stripper, 'out_file', convert2itk, 'reference_file')
```

Compute registration between the subject's structural and MNI template This is currently set to perform a very quick registration. However, the registration can be made significantly more accurate for cortical structures by increasing the number of iterations All parameters are set using the example from: <https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh>

```
reg = Node(ants.Registration(), name='antsRegister')
reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = ['Rigid', 'Affine', 'SyN']
reg.inputs.transform_parameters = [(0.1,), (0.1,), (0.2, 3.0, 0.0)]
reg.inputs.number_of_iterations = [[10000, 11110, 11110]] * 2 + [[100, 30, 20]]
reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = True
reg.inputs.initial_moving_transform_com = True
reg.inputs.metric = ['Mattes'] * 2 + [['Mattes', 'CC']]
reg.inputs.metric_weight = [1] * 2 + [[0.5, 0.5]]
reg.inputs.radius_or_number_of_bins = [32] * 2 + [[32, 4]]
reg.inputs.sampling_strategy = ['Regular'] * 2 + [[None, None]]
reg.inputs.sampling_percentage = [0.3] * 2 + [[None, None]]
reg.inputs.convergence_threshold = [1.e-8] * 2 + [-0.01]
reg.inputs.convergence_window_size = [20] * 2 + [5]
reg.inputs.smoothing_sigmas = [[4, 2, 1]] * 2 + [[1, 0.5, 0]]
reg.inputs.sigma_units = ['vox'] * 3
reg.inputs.shrink_factors = [[3, 2, 1]] * 2 + [[4, 2, 1]]
reg.inputs.use_estimate_learning_rate_once = [True] * 3
reg.inputs.use_histogram_matching = [False] * 2 + [True]
```

```
reg.inputs.winsorize_lower_quantile = 0.005
reg.inputs.winsorize_upper_quantile = 0.995
reg.inputs.float = True
reg.inputs.output_warped_image = 'output_warped_image.nii.gz'
reg.inputs.num_threads = 4
reg.plugin_args = {'qsub_args': '-pe orte 4',
                   'sbatch_args': '--mem=6G -c 4'}
register.connect(stripper, 'out_file', reg, 'moving_image')
register.connect(inputnode, 'target_image', reg, 'fixed_image')
```

Concatenate the affine and ants transforms into a list

```
pickfirst = lambda x: x[0]

merge = Node(Merge(2), iterfield=['in2'], name='mergexfm')
register.connect(convert2itk, 'itk_transform', merge, 'in2')
register.connect(reg, 'composite_transform', merge, 'in1')
```

Transform the mean image. First to anatomical and then to target

```
warpmean = Node(ants.ApplyTransforms(), name='warpmean')
warpmean.inputs.input_image_type = 0
warpmean.inputs.interpolation = 'Linear'
warpmean.inputs.invert_transform_flags = [False, False]
warpmean.inputs.terminal_output = 'file'
warpmean.inputs.args = '--float'
# warpmean.inputs.num_threads = 4
# warpmean.plugin_args = {'sbatch_args': '--mem=4G -c 4'}
```

Transform the remaining images. First to anatomical and then to target

```
warpall = pe.MapNode(ants.ApplyTransforms(),
                    iterfield=['input_image'],
                    name='warpall')
warpall.inputs.input_image_type = 0
warpall.inputs.interpolation = 'Linear'
warpall.inputs.invert_transform_flags = [False, False]
warpall.inputs.terminal_output = 'file'
warpall.inputs.args = '--float'
warpall.inputs.num_threads = 2
warpall.plugin_args = {'sbatch_args': '--mem=6G -c 2'}
```

Assign all the output files

```
register.connect(warpmean, 'output_image', outputnode, 'transformed_mean')
register.connect(warpall, 'output_image', outputnode, 'transformed_files')

register.connect(inputnode, 'target_image', warpmean, 'reference_image')
register.connect(inputnode, 'mean_image', warpmean, 'input_image')
register.connect(merge, 'out', warpmean, 'transforms')
register.connect(inputnode, 'target_image', warpall, 'reference_image')
register.connect(inputnode, 'source_files', warpall, 'input_image')
register.connect(merge, 'out', warpall, 'transforms')
```

Assign all the output files

```
register.connect(reg, 'warped_image', outputnode, 'anat2target')
register.connect(aparcxfm, 'transformed_file',
                outputnode, 'aparc')
register.connect(bbregister, 'out_fsl_file',
                outputnode, 'func2anat_transform')
```



```

register.connect(bregister, 'out_reg_file',
                 outputnode, 'out_reg_file')
register.connect(bregister, 'min_cost_file',
                 outputnode, 'min_cost_file')
register.connect(mean2anat_mask, 'mask_file',
                 outputnode, 'mean2anat_mask')
register.connect(reg, 'composite_transform',
                 outputnode, 'anat2target_transform')
register.connect(merge, 'out', outputnode, 'transforms')

return register

```

Get info for a given subject

```

def get_subjectinfo(subject_id, base_dir, task_id, model_id):
    """Get info for a given subject

    Parameters
    -----
    subject_id : string
        Subject identifier (e.g., sub001)
    base_dir : string
        Path to base directory of the dataset
    task_id : int
        Which task to process
    model_id : int
        Which model to process

    Returns
    -----
    run_ids : list of ints
        Run numbers
    conds : list of str
        Condition names
    TR : float
        Repetition time
    """
    from glob import glob
    import os
    import numpy as np
    condition_info = []
    cond_file = os.path.join(base_dir, 'models', 'model%03d' % model_id,
                             'condition_key.txt')
    with open(cond_file, 'rt') as fp:
        for line in fp:
            info = line.strip().split()
            condition_info.append([info[0], info[1], ' '.join(info[2:])])
    if len(condition_info) == 0:
        raise ValueError('No condition info found in %s' % cond_file)
    taskinfo = np.array(condition_info)
    n_tasks = len(np.unique(taskinfo[:, 0]))
    conds = []
    run_ids = []
    if task_id > n_tasks:
        raise ValueError('Task id %d does not exist' % task_id)
    for idx in range(n_tasks):
        taskidx = np.where(taskinfo[:, 0] == 'task%03d' % (idx + 1))
        conds.append([condition.replace(' ', '_') for condition

```

```

        in taskinfo[taskidx[0], 2])) # if 'junk' not in condition))
files = sorted(glob(os.path.join(base_dir,
                                subject_id,
                                'BOLD',
                                'task%03d_run*' % (idx + 1))))
runs = [int(val[-3:]) for val in files]
run_ids.insert(idx, runs)
json_info = os.path.join(base_dir, subject_id, 'BOLD',
                        'task%03d_run%03d' % (task_id, run_ids[task_id -
→1][0]),
                        'bold_scaninfo.json')
if os.path.exists(json_info):
    import json
    with open(json_info, 'rt') as fp:
        data = json.load(fp)
        TR = data['global']['const']['RepetitionTime'] / 1000.
else:
    task_scan_key = os.path.join(base_dir, subject_id, 'BOLD',
                                'task%03d_run%03d' % (task_id, run_ids[task_
→id - 1][0]),
                                'scan_key.txt')
    if os.path.exists(task_scan_key):
        TR = np.genfromtxt(task_scan_key) [1]
    else:
        TR = np.genfromtxt(os.path.join(base_dir, 'scan_key.txt')) [1]
return run_ids[task_id - 1], conds[task_id - 1], TR

```

Analyzes an open fmri dataset

```

def analyze_openfmri_dataset(data_dir, subject=None, model_id=None,
                             task_id=None, output_dir=None, subj_prefix='*',
                             hpcutoff=120., use_derivatives=True,
                             fwhm=6.0, subjects_dir=None, target=None):
    """Analyzes an open fmri dataset

    Parameters
    -----

    data_dir : str
        Path to the base data directory

    work_dir : str
        Nipype working directory (defaults to cwd)
    """

```

Load nipype workflows

```

preproc = create_featreg_preproc(whichvol='first')
modelfit = create_modelfit_workflow()
fixed_fx = create_fixed_effects_flow()
if subjects_dir:
    registration = create_fs_reg_workflow()
else:
    registration = create_reg_workflow()

```

Remove the plotting connection so that plot iterables don't propagate to the model stage

```

preproc.disconnect(preproc.get_node('plot_motion'), 'out_file',
                  preproc.get_node('outputspec'), 'motion_plots')

```

Set up openfmri data specific components

```

subjects = sorted([path.split(os.path.sep)[-1] for path in
                    glob(os.path.join(data_dir, subj_prefix))])

infosource = pe.Node(niu.IdentityInterface(fields=['subject_id',
                                                  'model_id',
                                                  'task_id']),
                    name='infosource')

if len(subject) == 0:
    infosource.iterables = [('subject_id', subjects),
                           ('model_id', [model_id]),
                           ('task_id', task_id)]
else:
    infosource.iterables = [('subject_id',
                             [subjects[subjects.index(subj)] for subj in
                             ↪subject]),
                           ('model_id', [model_id]),
                           ('task_id', task_id)]

subjinfo = pe.Node(niu.Function(input_names=['subject_id', 'base_dir',
                                             'task_id', 'model_id'],
                                output_names=['run_id', 'conds', 'TR'],
                                function=get_subjectinfo),
                  name='subjectinfo')
subjinfo.inputs.base_dir = data_dir

```

Return data components as anat, bold and behav

```

contrast_file = os.path.join(data_dir, 'models', 'model%03d' % model_id,
                              'task_contrasts.txt')
has_contrast = os.path.exists(contrast_file)
if has_contrast:
    datasource = pe.Node(nio.DataGrabber(infields=['subject_id', 'run_id',
                                                  'task_id', 'model_id'],
                                         outfields=['anat', 'bold', 'behav',
                                                    'contrasts']),
                        name='datasource')
else:
    datasource = pe.Node(nio.DataGrabber(infields=['subject_id', 'run_id',
                                                  'task_id', 'model_id'],
                                         outfields=['anat', 'bold', 'behav']),
                        name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '*'

if has_contrast:
    datasource.inputs.field_template = {'anat': '%s/anatomy/T1_001.nii.gz',
                                       'bold': '%s/BOLD/task%03d_r*/bold.nii.gz',
                                       'behav': ('%s/model/model%03d/onsets/task
↪%03d_'
                                              'run%03d/cond*.txt'),
                                       'contrasts': ('models/model%03d/'
                                                    'task_contrasts.txt')}
    datasource.inputs.template_args = {'anat': [['subject_id']],
                                       'bold': [['subject_id', 'task_id']],
                                       'behav': [['subject_id', 'model_id',
                                                'task_id', 'run_id']],
                                       'contrasts': [['model_id']]}

```

```

else:
    datasource.inputs.field_template = {'anat': '%s/anatomy/T1_001.nii.gz',
                                        'bold': '%s/BOLD/task%03d_r*/bold.nii.gz',
                                        'behav': ('%s/model/model%03d/onsets/task
↳%03d_
                                                'run%03d/cond*.txt'))}

    datasource.inputs.template_args = {'anat': [['subject_id']],
                                       'bold': [['subject_id', 'task_id']],
                                       'behav': [['subject_id', 'model_id',
                                                'task_id', 'run_id']]}

datasource.inputs.sort_filelist = True

```

Create meta workflow

```

wf = pe.Workflow(name='openfmri')
wf.connect(infosource, 'subject_id', subjinfo, 'subject_id')
wf.connect(infosource, 'model_id', subjinfo, 'model_id')
wf.connect(infosource, 'task_id', subjinfo, 'task_id')
wf.connect(infosource, 'subject_id', datasource, 'subject_id')
wf.connect(infosource, 'model_id', datasource, 'model_id')
wf.connect(infosource, 'task_id', datasource, 'task_id')
wf.connect(subjinfo, 'run_id', datasource, 'run_id')
wf.connect([(datasource, preproc, [('bold', 'inputspec.func')]),
           ])

def get_highpass(TR, hpcutoff):
    return hpcutoff / (2. * TR)
gethighpass = pe.Node(niu.Function(input_names=['TR', 'hpcutoff'],
                                   output_names=['highpass'],
                                   function=get_highpass),
                      name='gethighpass')
wf.connect(subjinfo, 'TR', gethighpass, 'TR')
wf.connect(gethighpass, 'highpass', preproc, 'inputspec.highpass')

```

Setup a basic set of contrasts, a t-test per condition

```

def get_contrasts(contrast_file, task_id, conds):
    import numpy as np
    import os
    contrast_def = []
    if os.path.exists(contrast_file):
        with open(contrast_file, 'rt') as fp:
            contrast_def.extend([np.array(row.split()) for row in fp.readlines()
↳if row.strip()])
    contrasts = []
    for row in contrast_def:
        if row[0] != 'task%03d' % task_id:
            continue
        con = [row[1], 'T', ['cond%03d' % (i + 1) for i in range(len(conds))],
              row[2:].astype(float).tolist()]
        contrasts.append(con)
    # add auto contrasts for each column
    for i, cond in enumerate(conds):
        con = [cond, 'T', ['cond%03d' % (i + 1)], [1]]
        contrasts.append(con)
    return contrasts

contrastgen = pe.Node(niu.Function(input_names=['contrast_file',

```

```

        'task_id', 'conds'],
        output_names=['contrasts'],
        function=get_contrasts),
        name='contrastgen')

art = pe.MapNode(interface=ra.ArtifactDetect(use_differences=[True, False],
        use_norm=True,
        norm_threshold=1,
        zintensity_threshold=3,
        parameter_source='FSL',
        mask_type='file'),
        iterfield=['realigned_files', 'realignment_parameters',
        'mask_file'],
        name="art")

modelspec = pe.Node(interface=model.SpecifyModel(),
        name="modelspec")
modelspec.inputs.input_units = 'secs'

def check_behav_list(behav, run_id, conds):
    import numpy as np
    num_conds = len(conds)
    if isinstance(behav, (str, bytes)):
        behav = [behav]
    behav_array = np.array(behav).flatten()
    num_elements = behav_array.shape[0]
    return behav_array.reshape(int(num_elements / num_conds),
        num_conds).tolist()

reshape_behav = pe.Node(niu.Function(input_names=['behav', 'run_id', 'conds'],
        output_names=['behav'],
        function=check_behav_list),
        name='reshape_behav')

wf.connect(subjinfo, 'TR', modelspec, 'time_repetition')
wf.connect(datasource, 'behav', reshape_behav, 'behav')
wf.connect(subjinfo, 'run_id', reshape_behav, 'run_id')
wf.connect(subjinfo, 'conds', reshape_behav, 'conds')
wf.connect(reshape_behav, 'behav', modelspec, 'event_files')

wf.connect(subjinfo, 'TR', modelfit, 'inputspec.interscan_interval')
wf.connect(subjinfo, 'conds', contrastgen, 'conds')
if has_contrast:
    wf.connect(datasource, 'contrasts', contrastgen, 'contrast_file')
else:
    contrastgen.inputs.contrast_file = ''
wf.connect(infosource, 'task_id', contrastgen, 'task_id')
wf.connect(contrastgen, 'contrasts', modelfit, 'inputspec.contrasts')

wf.connect([(preproc, art, [('outputspec.motion_parameters',
        'realignment_parameters'),
        ('outputspec.realigned_files',
        'realigned_files'),
        ('outputspec.mask', 'mask_file')]),
        (preproc, modelspec, [('outputspec.highpassed_files',
        'functional_runs'),
        ('outputspec.motion_parameters',
        'realignment_parameters')])],

```

```

        (art, modelspec, [('outlier_files', 'outlier_files')]),
        (modelspec, modelfit, [('session_info',
                                'inputspec.session_info')]),
        (preproc, modelfit, [('outputspec.highpassed_files',
                                'inputspec.functional_data')])
    ])

# Compute TSNR on realigned data regressing polynomials upto order 2
tsnr = MapNode(TSNR(regress_poly=2), iterfield=['in_file'], name='tsnr')
wf.connect(preproc, "outputspec.realigned_files", tsnr, "in_file")

# Compute the median image across runs
calc_median = Node(Function(input_names=['in_files'],
                             output_names=['median_file'],
                             function=median,
                             imports=imports),
                    name='median')
wf.connect(tsnr, 'detrended_file', calc_median, 'in_files')

```

Reorder the copes so that now it combines across runs

```

def sort_copes(copes, varcopes, contrasts):
    import numpy as np
    if not isinstance(copes, list):
        copes = [copes]
        varcopes = [varcopes]
    num_copes = len(contrasts)
    n_runs = len(copes)
    all_copes = np.array(copes).flatten()
    all_varcopes = np.array(varcopes).flatten()
    outcopes = all_copes.reshape(int(len(all_copes) / num_copes),
                                num_copes).T.tolist()
    outvarcopes = all_varcopes.reshape(int(len(all_varcopes) / num_copes),
                                       num_copes).T.tolist()
    return outcopes, outvarcopes, n_runs

cope_sorter = pe.Node(niu.Function(input_names=['copes', 'varcopes',
                                                'contrasts'],
                                   output_names=['copes', 'varcopes',
                                                  'n_runs'],
                                   function=sort_copes),
                      name='cope_sorter')

pickfirst = lambda x: x[0]

wf.connect(contrastgen, 'contrasts', cope_sorter, 'contrasts')
wf.connect([(preproc, fixed_fx, [('outputspec.mask', pickfirst),
                                'flameo.mask_file'])],
            (modelfit, cope_sorter, [('outputspec.copes', 'copes')]),
            (modelfit, cope_sorter, [('outputspec.varcopes', 'varcopes')]),
            (cope_sorter, fixed_fx, [('copes', 'inputspec.copes'),
                                    ('varcopes', 'inputspec.varcopes'),
                                    ('n_runs', 'l2model.num_copes')]),
            (modelfit, fixed_fx, [('outputspec.dof_file',
                                'inputspec.dof_files')]),
            ])

```

```

wf.connect(calc_median, 'median_file', registration, 'inputspec.mean_image')
if subjects_dir:
    wf.connect(infosource, 'subject_id', registration, 'inputspec.subject_id')
    registration.inputs.inputs.spec.subjects_dir = subjects_dir
    registration.inputs.inputs.spec.target_image = fsl.Info.standard_image('MNI152_
↪T1_2mm_brain.nii.gz')
    if target:
        registration.inputs.inputs.spec.target_image = target
else:
    wf.connect(datasource, 'anat', registration, 'inputspec.anatomical_image')
    registration.inputs.inputs.spec.target_image = fsl.Info.standard_image('MNI152_
↪T1_2mm.nii.gz')
    registration.inputs.inputs.spec.target_image_brain = fsl.Info.standard_image(
↪'MNI152_T1_2mm_brain.nii.gz')
    registration.inputs.inputs.spec.config_file = 'T1_2_MNI152_2mm'

def merge_files(copes, varcopes, zstats):
    out_files = []
    splits = []
    out_files.extend(copes)
    splits.append(len(copes))
    out_files.extend(varcopes)
    splits.append(len(varcopes))
    out_files.extend(zstats)
    splits.append(len(zstats))
    return out_files, splits

mergefunc = pe.Node(niu.Function(input_names=['copes', 'varcopes',
                                             'zstats'],
                                output_names=['out_files', 'splits'],
                                function=merge_files),
                    name='merge_files')
wf.connect([(fixed_fx.get_node('outputspec'), mergefunc,
    [(['copes', 'copes'],
      ['varcopes', 'varcopes'],
      ['zstats', 'zstats'],
      ])]))
wf.connect(mergefunc, 'out_files', registration, 'inputspec.source_files')

def split_files(in_files, splits):
    copes = in_files[:splits[0]]
    varcopes = in_files[splits[0]:(splits[0] + splits[1])]
    zstats = in_files[(splits[0] + splits[1]):]
    return copes, varcopes, zstats

splitfunc = pe.Node(niu.Function(input_names=['in_files', 'splits'],
                                   output_names=['copes', 'varcopes',
                                                  'zstats'],
                                   function=split_files),
                    name='split_files')
wf.connect(mergefunc, 'splits', splitfunc, 'splits')
wf.connect(registration, 'outputspec.transformed_files',
            splitfunc, 'in_files')

if subjects_dir:
    get_roi_mean = pe.MapNode(fs.SegStats(default_color_table=True),
                              iterfield=['in_file'], name='get_aparc_means')
    get_roi_mean.inputs.avgwf_txt_file = True

```

```
wf.connect(fixed_fx.get_node('outputspec'), 'copes', get_roi_mean, 'in_file')
wf.connect(registration, 'outputspec.aparc', get_roi_mean, 'segmentation_file
→')

get_roi_tsnr = pe.MapNode(fs.SegStats(default_color_table=True),
                           iterfield=['in_file'], name='get_aparc_tsnr')
get_roi_tsnr.inputs.avgwf_txt_file = True
wf.connect(tsnr, 'tsnr_file', get_roi_tsnr, 'in_file')
wf.connect(registration, 'outputspec.aparc', get_roi_tsnr, 'segmentation_file
→')
```

Connect to a datasink

```
def get_subs(subject_id, conds, run_id, model_id, task_id):
    subs = [('_subject_id_%s_' % subject_id, '')]
    subs.append(('_model_id_%d' % model_id, 'model%03d' % model_id))
    subs.append(('task_id_%d/' % task_id, '/task%03d_' % task_id))
    subs.append(('bold_dtype_mcf_mask_smooth_mask_gms_tempfilt_mean_warp',
                  'mean'))
    subs.append(('bold_dtype_mcf_mask_smooth_mask_gms_tempfilt_mean_flirt',
                  'affine'))

    for i in range(len(conds)):
        subs.append(('_flameo%d/cope1.' % i, 'cope%02d.' % (i + 1)))
        subs.append(('_flameo%d/varcope1.' % i, 'varcope%02d.' % (i + 1)))
        subs.append(('_flameo%d/zstat1.' % i, 'zstat%02d.' % (i + 1)))
        subs.append(('_flameo%d/tstat1.' % i, 'tstat%02d.' % (i + 1)))
        subs.append(('_flameo%d/res4d.' % i, 'res4d%02d.' % (i + 1)))
        subs.append(('warpall%d/cope1_warp.' % i,
                      'cope%02d.' % (i + 1)))
        subs.append(('warpall%d/varcope1_warp.' % (len(conds) + i),
                      'varcope%02d.' % (i + 1)))
        subs.append(('warpall%d/zstat1_warp.' % (2 * len(conds) + i),
                      'zstat%02d.' % (i + 1)))
        subs.append(('warpall%d/cope1_trans.' % i,
                      'cope%02d.' % (i + 1)))
        subs.append(('warpall%d/varcope1_trans.' % (len(conds) + i),
                      'varcope%02d.' % (i + 1)))
        subs.append(('warpall%d/zstat1_trans.' % (2 * len(conds) + i),
                      'zstat%02d.' % (i + 1)))
        subs.append(('__get_aparc_means%d/' % i, '/cope%02d_' % (i + 1)))

    for i, run_num in enumerate(run_id):
        subs.append(('__get_aparc_tsnr%d/' % i, '/run%02d_' % run_num))
        subs.append(('__art%d/' % i, '/run%02d_' % run_num))
        subs.append(('__dilatemask%d/' % i, '/run%02d_' % run_num))
        subs.append(('__realign%d/' % i, '/run%02d_' % run_num))
        subs.append(('__modelgen%d/' % i, '/run%02d_' % run_num))
        subs.append(('model%03d/task%03d/' % (model_id, task_id), '/'))
        subs.append(('model%03d/task%03d_' % (model_id, task_id), '/'))
        subs.append(('bold_dtype_mcf_bet_thresh_dil', '_mask'))
        subs.append(('output_warped_image', '_anat2target'))
        subs.append(('median_flirt_brain_mask', 'median_brain_mask'))
        subs.append(('median_bbgreg_brain_mask', 'median_brain_mask'))
    return subs

subsgen = pe.Node(niu.Function(input_names=['subject_id', 'conds', 'run_id',
                                             'model_id', 'task_id'],
```



```

        output_names=['substitutions'],
        function=get_subs),
        name='subsgen')
wf.connect(subjinfo, 'run_id', subsgen, 'run_id')

datasink = pe.Node(interface=nio.DataSink(),
                    name="datasink")
wf.connect(infosource, 'subject_id', datasink, 'container')
wf.connect(infosource, 'subject_id', subsgen, 'subject_id')
wf.connect(infosource, 'model_id', subsgen, 'model_id')
wf.connect(infosource, 'task_id', subsgen, 'task_id')
wf.connect(contrastgen, 'contrasts', subsgen, 'conds')
wf.connect(subsgen, 'substitutions', datasink, 'substitutions')
wf.connect([(fixed_fx.get_node('outputspec'), datasink,
        [ ('res4d', 'res4d'),
          ('copes', 'copes'),
          ('varcopes', 'varcopes'),
          ('zstats', 'zstats'),
          ('tstats', 'tstats')])
        ]))
wf.connect([(modelfit.get_node('modelgen'), datasink,
        [ ('design_cov', 'qa.model'),
          ('design_image', 'qa.model.@matrix_image'),
          ('design_file', 'qa.model.@matrix'),
          ]))
wf.connect([(preproc, datasink, [ ('outputspec.motion_parameters',
        'qa.motion'),
        ('outputspec.motion_plots',
        'qa.motion.plots'),
        ('outputspec.mask', 'qa.mask')])])
wf.connect(registration, 'outputspec.mean2anat_mask', datasink, 'qa.mask.mean2anat
↪')
wf.connect(art, 'norm_files', datasink, 'qa.art.@norm')
wf.connect(art, 'intensity_files', datasink, 'qa.art.@intensity')
wf.connect(art, 'outlier_files', datasink, 'qa.art.@outlier_files')
wf.connect(registration, 'outputspec.anat2target', datasink, 'qa.anat2target')
wf.connect(tsnr, 'tsnr_file', datasink, 'qa.tsnr.@map')
if subjects_dir:
    wf.connect(registration, 'outputspec.min_cost_file', datasink, 'qa.mincost')
    wf.connect([(get_roi_tsnr, datasink, [ ('avgwf_txt_file', 'qa.tsnr'),
        ('summary_file', 'qa.tsnr.@summary')])])
    wf.connect([(get_roi_mean, datasink, [ ('avgwf_txt_file', 'copes.roi'),
        ('summary_file', 'copes.roi.@summary
↪')])])
wf.connect([(splitfunc, datasink,
        [ ('copes', 'copes.mni'),
          ('varcopes', 'varcopes.mni'),
          ('zstats', 'zstats.mni'),
          ]))
wf.connect(calc_median, 'median_file', datasink, 'mean')
wf.connect(registration, 'outputspec.transformed_mean', datasink, 'mean.mni')
wf.connect(registration, 'outputspec.func2anat_transform', datasink, 'xfm.
↪mean2anat')
wf.connect(registration, 'outputspec.anat2target_transform', datasink, 'xfm.
↪anat2target')

```

Set processing parameters

```

preproc.inputs.inputs.spec.fwhm = fwhm
gethighpass.inputs.hpcutoff = hpcutoff
modelspec.inputs.high_pass_filter_cutoff = hpcutoff
modelfit.inputs.inputs.spec.bases = {'dgamma': {'derivs': use_derivatives}}
modelfit.inputs.inputs.spec.model_serial_correlations = True
modelfit.inputs.inputs.spec.film_threshold = 1000

datasink.inputs.base_directory = output_dir
return wf

```

The following functions run the whole workflow.

```

if __name__ == '__main__':
    import argparse
    defstr = ' (default %(default)s)'
    parser = argparse.ArgumentParser(prog='fmri_openfmri.py',
                                     description=__doc__)
    parser.add_argument('-d', '--datasetdir', required=True)
    parser.add_argument('-s', '--subject', default=[],
                        nargs='+', type=str,
                        help="Subject name (e.g. 'sub001')")
    parser.add_argument('-m', '--model', default=1,
                        help="Model index" + defstr)
    parser.add_argument('-x', '--subjectprefix', default='sub*',
                        help="Subject prefix" + defstr)
    parser.add_argument('-t', '--task', default=1, # nargs='+',
                        type=int, help="Task index" + defstr)
    parser.add_argument('--hpfilter', default=120.,
                        type=float, help="High pass filter cutoff (in secs)" +
↳ defstr)
    parser.add_argument('--fwhm', default=6.,
                        type=float, help="Spatial FWHM" + defstr)
    parser.add_argument('--derivatives', action="store_true",
                        help="Use derivatives" + defstr)
    parser.add_argument("-o", "--output_dir", dest="outdir",
                        help="Output directory base")
    parser.add_argument("-w", "--work_dir", dest="work_dir",
                        help="Output directory base")
    parser.add_argument("-p", "--plugin", dest="plugin",
                        default='Linear',
                        help="Plugin to use")
    parser.add_argument("--plugin_args", dest="plugin_args",
                        help="Plugin arguments")
    parser.add_argument("--sd", dest="subjects_dir",
                        help="FreeSurfer subjects directory (if available)")
    parser.add_argument("--target", dest="target_file",
                        help=("Target in MNI space. Best to use the MindBoggle "
                              "template - only used with FreeSurfer "
                              "OASIS-30_Atropos_template_in_MNI152_2mm.nii.gz"))
    args = parser.parse_args()
    outdir = args.outdir
    work_dir = os.getcwd()
    if args.work_dir:
        work_dir = os.path.abspath(args.work_dir)
    if outdir:
        outdir = os.path.abspath(outdir)
    else:
        outdir = os.path.join(work_dir, 'output')

```

```
outdir = os.path.join(outdir, 'model%02d' % int(args.model),
                          'task%03d' % int(args.task))
derivatives = args.derivatives
if derivatives is None:
    derivatives = False
wf = analyze_openfmri_dataset(data_dir=os.path.abspath(args.datasetdir),
                             subject=args.subject,
                             model_id=int(args.model),
                             task_id=[int(args.task)],
                             subj_prefix=args.subjectprefix,
                             output_dir=outdir,
                             hpcutoff=args.hpfilter,
                             use_derivatives=derivatives,
                             fwhm=args.fwhm,
                             subjects_dir=args.subjects_dir,
                             target=args.target_file)

# wf.config['execution']['remove_unnecessary_outputs'] = False

wf.base_dir = work_dir
if args.plugin_args:
    wf.run(args.plugin, plugin_args=eval(args.plugin_args))
else:
    wf.run(args.plugin)
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipyte source distribution under the `examples` directory.

fMRI: surface smooth - FreeSurfer, SPM

This tutorial illustrates how to perform surface-based smoothing of cortical data using [FreeSurfer](#) and then perform firstlevel model and contrast estimation using [SPM](#). A surface-based second level glm illustrates the use of spherical registration and freesurfer's glm functions.

Preparing environment

Step 0

In order to run this tutorial you need to have [SPM](#) and [FreeSurfer](#) tools installed and accessible from matlab/command line. Check by calling `mri_info` from the command line.

Step 1

Link the *fsaverage* directory for your freesurfer distribution. To do this type:

```
cd nipy-tutorial/fsdata
ln -s $FREESURFER_HOME/subjects/fsaverage
cd ..
```

Defining the workflow

```
from __future__ import print_function
from builtins import str
from builtins import range

import os                                     # system functions

import nipy.algorithms.modelgen as model     # model generation
import nipy.algorithms.rapidart as ra        # artifact detection
import nipy.interfaces.freesurfer as fs      # freesurfer
import nipy.interfaces.io as nio             # i/o routines
import nipy.interfaces.matlab as mlab        # how to run matlab
import nipy.interfaces.spm as spm            # spm
```

```
import nipyype.interfaces.utility as util      # utility
import nipyype.pipeline.engine as pe          # pypeline engine
```

Preliminaries

Set any package specific configuration.

Setting the subjects directory and the appropriate matlab command to use. if you want to use a different spm version/path, it should also be entered here.

These are currently being set at the class level, so every node will inherit these settings. However, these can also be changed or set for an individual node.

```
# Tell freesurfer what subjects directory to use
subjects_dir = os.path.abspath('fsdata')
fs.FSCommand.set_default_subjects_dir(subjects_dir)

# Set the way matlab should be called
mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
# If SPM is not in your MATLAB path you should add it here
mlab.MatlabCommand.set_default_paths('/software/spm8')
```

Setup preprocessing workflow

```
preproc = pe.Workflow(name='preproc')
```

Use `nipyype.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```
realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True
```

Use `nipyype.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```
art = pe.Node(interface=ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```

Use `nipyype.interfaces.freesurfer.BBRegister` to coregister the mean functional image generated by `realign` to the subjects' surfaces.

```
surfregister = pe.Node(interface=fs.BBRegister(), name='surfregister')
surfregister.inputs.init = 'fsl'
surfregister.inputs.contrast_type = 't2'
```

Use `nipyype.interfaces.io.FreeSurferSource` to retrieve various image files that are automatically generated by the recon-all process.

```
FreeSurferSource = pe.Node(interface=nio.FreeSurferSource(), name='fssource')
```

Use `nipyype.interfaces.freesurfer.ApplyVolTransform` to convert the brainmask generated by `freesurfer` into the realigned functional space.

```
ApplyVolTransform = pe.Node(interface=fs.ApplyVolTransform(),
                             name='applyreg')
ApplyVolTransform.inputs.inverse = True
```

Use `nipyne.interfaces.freesurfer.Binarize` to extract a binary brain mask.

```
Threshold = pe.Node(interface=fs.Binarize(), name='threshold')
Threshold.inputs.min = 10
Threshold.inputs.out_type = 'nii'
```

Two different types of functional data smoothing are performed in this workflow. The volume smoothing option performs a standard SPM smoothen. using `nipyne.interfaces.spm.Smooth`. In addition, we use a smoothing routine from freesurfer (`nipyne.interfaces.freesurfer.Binarize`) to project the functional data from the volume to the subjects' surface, smooth it on the surface and fit it back into the volume forming the cortical ribbon. The projection uses the average value along a "cortical column". In addition to the surface smoothing, the rest of the volume is smoothed with a 3d gaussian kernel.

Note: It is very important to note that the projection to the surface takes a 3d manifold to a 2d manifold. Hence the reverse projection, simply fills the thickness of cortex with the smoothed data. The smoothing is not performed in a depth specific manner. The output of this branch should only be used for surface-based analysis and visualization.

```
volsmooth = pe.Node(interface=spm.Smooth(), name="volsmooth")
surfsmooth = pe.MapNode(interface=fs.Smooth(proj_frac_avg=(0, 1, 0.1)), name=
    ↪ "surfsmooth",
                        iterfield=['in_file'])
```

We connect up the different nodes to implement the preprocessing workflow.

```
preproc.connect([(realign, surfregister, [('mean_image', 'source_file')]),
                 (FreeSurferSource, ApplyVolTransform, [('brainmask', 'target_file'
    ↪ ')]),
                 (surfregister, ApplyVolTransform, [('out_reg_file', 'reg_file'
    ↪ ')]),
                 (realign, ApplyVolTransform, [('mean_image', 'source_file')]),
                 (ApplyVolTransform, Threshold, [('transformed_file', 'in_file'
    ↪ ')]),
                 (realign, art, [('realignment_parameters', 'realignment_
    ↪ parameters'),
                                ('realigned_files', 'realigned_files')]),
                 (Threshold, art, [('binary_file', 'mask_file')]),
                 (realign, volsmooth, [('realigned_files', 'in_files')]),
                 (realign, surfsmooth, [('realigned_files', 'in_file')]),
                 (surfregister, surfsmooth, [('out_reg_file', 'reg_file')]),
                 ])
```

Set up volume analysis workflow

```
volanalysis = pe.Workflow(name='volanalysis')
```

Generate SPM-specific design information using `nipyne.interfaces.spm.SpecifyModel`.

```
modelspec = pe.Node(interface=model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = True
```

Generate a first level SPM.mat file for analysis `nipyne.interfaces.spm.Level1Design`.

```
level1design = pe.Node(interface=spm.Level1Design(), name="level1design")
level1design.inputs.bases = {'hrf': {'derivs': [0, 0]}}
```

Use `nipyne.interfaces.spm.EstimateModel` to determine the parameters of the model.

```
levelleestimate = pe.Node(interface=spm.EstimateModel(), name="levelleestimate")
levelleestimate.inputs.estimate_method = {'Classical': 1}
```

Use `nipyne.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.

```
contrastestimate = pe.Node(interface=spm.EstimateContrast(), name=
↳ "contrastestimate")

volanalysis.connect([(modelspec, levelldesign, [('session_info', 'session_info
↳ ')]),
                    (levelldesign, levelleestimate, [('spm_mat_file', 'spm_mat_
↳ file')]),
                    (levelleestimate, contrastestimate, [('spm_mat_file', 'spm_
↳ mat_file'),
                                                         ('beta_images', 'beta_
↳ images'),
                                                         ('residual_image',
↳ 'residual_image')]),
                    ])
```

Set up surface analysis workflow

We simply clone the volume analysis workflow.

```
surfanalysis = volanalysis.clone(name='surfanalysis')
```

Set up volume normalization workflow

The volume analysis is performed in individual space. Therefore, post analysis we normalize the contrast images to MNI space.

```
volnorm = pe.Workflow(name='volnormconimages')
```

Use `nipyne.interfaces.freesurfer.MRISConvert` to convert the brainmask, an mgz file and the contrast images (nifti-1 img/hdr pairs), to single volume nifti images.

```
convert = pe.Node(interface=fs.MRISConvert(out_type='nii'), name='convert2nii')
convert2 = pe.MapNode(interface=fs.MRISConvert(out_type='nii'),
                      iterfield=['in_file'],
                      name='convertimg2nii')
```

Use `nipyne.interfaces.spm.Segment` to segment the structural image and generate the transformation file to MNI space.

Note: Segment takes longer than usual because the nose is wrapped behind the head in the structural image.

```
segment = pe.Node(interface=spm.Segment(), name='segment')
```

Use `nipyne.interfaces.freesurfer.ApplyVolTransform` to convert contrast images into freesurfer space.

```
normwreg = pe.MapNode(interface=fs.ApplyVolTransform(),
                      iterfield=['source_file'],
                      name='applyreg2con')
```

Use `nipyne.interfaces.spm.Normalize` to normalize the contrast images to MNI space


```
normalize = pe.Node(interface=spm.Normalize(jobtype='write'),
                    name='norm2mni')
```

Connect up the volume normalization components

```
volnorm.connect([(convert, segment, [('out_file', 'data')]),
                 (convert2, normwreg, [('out_file', 'source_file')]),
                 (segment, normalize, [('transformation_mat', 'parameter_file')]),
                 (normwreg, normalize, [('transformed_file', 'apply_to_files')]),
                 ])
```

Preproc + Analysis + VolumeNormalization workflow

Connect up the lower level workflows into an integrated analysis. In addition, we add an input node that specifies all the inputs needed for this workflow. Thus, one can import this workflow and connect it to their own data sources. An example with the nifti-tutorial data is provided below.

For this workflow the only necessary inputs are the functional images, a freesurfer subject id corresponding to recon-all processed data, the session information for the functional runs and the contrasts to be evaluated.

```
inputnode = pe.Node(interface=util.IdentityInterface(fields=[ 'func',
                                                            'subject_id',
                                                            'session_info',
                                                            'contrasts']),
                    name='inputnode')
```

Connect the components into an integrated workflow.

```
l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect([(inputnode, preproc, [('func', 'realign.in_files',
                                           ('subject_id', 'surfregister.subject_id',
                                           ↪'),
                                           ('subject_id', 'fssource.subject_id',
                                           ↪)],
                    (inputnode, volanalysis, [('session_info', 'modelspec.subject_
↪info'),
                                           ('contrasts', 'contrastestimate.
↪contrasts')]),
                    (inputnode, surfanalysis, [('session_info', 'modelspec.
↪subject_info'),
                                           ('contrasts', 'contrastestimate.
↪contrasts')]),
                    ])
```

attach volume and surface model specification and estimation components

```
l1pipeline.connect([(preproc, volanalysis, [('realign.realignment_parameters',
                                              'modelspec.realignment_parameters'),
                                              ('volsmooth.smoothed_files',
                                              'modelspec.functional_runs'),
                                              ('art.outlier_files',
                                              'modelspec.outlier_files'),
                                              ('threshold.binary_file',
                                              'level1design.mask_image')]),
                    (preproc, surfanalysis, [('realign.realignment_parameters',
                                              'modelspec.realignment_parameters'),
                                              ('surfsmooth.smoothed_file',
                                              'modelspec.functional_runs'),
                                              ('art.outlier_files',
                                              'modelspec.outlier_files'),
```

```

                                ('threshold.binary_file',
                                'level1design.mask_image']])

    ])

# attach volume contrast normalization components
l1pipeline.connect([(preproc, volnorm, [('fsssource.orig', 'convert2nii.in_file'),
                                ('surfregister.out_reg_file',
                                ↪ 'applyreg2con.reg_file'),
                                ('fsssource.orig', 'applyreg2con.target_
                                ↪ file')])],
                    (volanalysis, volnorm, [('contrastestimate.con_images',
                                'convertimg2nii.in_file'),
                                ]))

    ])

```

Data specific components

The nipy tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume named struct.nii.

Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline.

In the example below, run ‘f3’ is of type ‘func’ and gets mapped to a nifti filename through a template ‘%s.nii’. So ‘f3’ would become ‘f3.nii’.

```

# Specify the location of the data.
data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1', 's3']
# Map field names to individual subject runs.
info = dict(func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],
            struct=[['subject_id', 'struct']])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                    ↪ name="infosource")

```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipy.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipy.pipeline.NodeWrapper` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```

datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

```

Set preprocessing parameters

```
l1pipeline.inputs.preproc.fssource.subjects_dir = subjects_dir
l1pipeline.inputs.preproc.volsmooth.fwhm = 4
l1pipeline.inputs.preproc.surfsmooth.surface_fwhm = 5
l1pipeline.inputs.preproc.surfsmooth.vol_fwhm = 4
```

Experimental paradigm specific components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the `nipyne.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant.

```
def subjectinfo(subject_id):
    from nipyne.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: %s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r,
                      Bunch(conditions=names,
                           onsets=deepcopy(onsets),
                           durations=[15 for s in names],
                           ))
    return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]]. The condition names must match the *names* listed in the *subjectinfo* function described above.

```
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]
```

Set up node specific inputs

We replicate the `modelspec` parameters separately for the surface- and volume-based analysis.

```
modelspecref = l1pipeline.inputs.volanalysis.modelspec
modelspecref.input_units = 'secs'
modelspecref.time_repetition = 3.
modelspecref.high_pass_filter_cutoff = 120

modelspecref = l1pipeline.inputs.surfanalysis.modelspec
modelspecref.input_units = 'secs'
modelspecref.time_repetition = 3.
modelspecref.high_pass_filter_cutoff = 120

l1designref = l1pipeline.inputs.volanalysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition

l1designref = l1pipeline.inputs.surfanalysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition
```

```
l1pipeline.inputs.inputnode.contrasts = contrasts
```

Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the `nipyype.pipeline.engine.Workflow` to create a graph-based execution pipeline for first level analysis.

```
level1 = pe.Workflow(name="level1")
level1.base_dir = os.path.abspath('volsurf_tutorial/workingdir')

level1.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                (datasource, l1pipeline, [('func', 'inputnode.func')]),
                (infosource, l1pipeline, [('subject_id', 'inputnode.subject_id'),
                                          (('subject_id', subjectinfo),
                                           'inputnode.session_info')]),
                ])
```

Store the output

Create a datasink node to store the contrast images and registration info

```
datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('volsurf_tutorial/l1out')
datasink.inputs.substitutions = []

def getsubs(subject_id):
    subs = [('subject_id_%s/' % subject_id, '')]
    return subs

# store relevant outputs from various stages of the 1st level analysis
level1.connect([(infosource, datasink, [('subject_id', 'container'),
                                         (('subject_id', getsubs), 'substitutions')
                                         ]),
                (l1pipeline, datasink, [('surfanalysis.contrastestimate.con_images
→', 'contrasts'),
                                         ('preproc.surfregister.out_reg_file',
→'registrations'),
                                         ]),
                ])
```

Run the analysis pipeline and also create a dot+png (if graphviz is available) that visually represents the workflow.

```
if __name__ == '__main__':
    level1.run()
    level1.write_graph(graph2use='flat')
```

Level2 surface-based pipeline

Create a level2 workflow

```
l2flow = pe.Workflow(name='l2out')
l2flow.base_dir = os.path.abspath('volsurf_tutorial')
```

Setup a dummy node to iterate over contrasts and hemispheres

```
l2inputnode = pe.Node(interface=util.IdentityInterface(fields=['contrasts',
                                                             'hemi']),
                      name='inputnode')
l2inputnode.iterables = [('contrasts', list(range(1, len(contrasts) + 1))),
                        ('hemi', ['lh', 'rh'])]
```

Use a datagrabber node to collect contrast images and registration files

```
l2source = pe.Node(interface=nio.DataGrabber(infields=['con_id'],
                                              outfields=['con', 'reg']),
                  name='l2source')
l2source.inputs.base_directory = os.path.abspath('volsurf_tutorial/11out')
l2source.inputs.template = '*'
l2source.inputs.field_template = dict(con='*/contrasts/con_%04d.img',
                                     reg='*/registrations/*.dat')
l2source.inputs.template_args = dict(con=[['con_id']], reg=[[]])
l2source.inputs.sort_filelist = True

l2flow.connect(l2inputnode, 'contrasts', l2source, 'con_id')
```

Merge contrast images and registration files

```
mergenode = pe.Node(interface=util.Merge(2, axis='hstack'),
                  name='merge')

def ordersubjects(files, subj_list):
    outlist = []
    for s in subj_list:
        for f in files:
            if '/%s/' % s in f:
                outlist.append(f)
            continue
    print(outlist)
    return outlist

l2flow.connect(l2source, ('con', ordersubjects, subject_list), mergenode, 'in1')
l2flow.connect(l2source, ('reg', ordersubjects, subject_list), mergenode, 'in2')
```

Concatenate contrast images projected to fsaverage

```
l2concat = pe.Node(interface=fs.MRISPreproc(), name='concat')
l2concat.inputs.target = 'fsaverage'
l2concat.inputs.fwhm = 5

def list2tuple(listoflist):
    return [tuple(x) for x in listoflist]

l2flow.connect(l2inputnode, 'hemi', l2concat, 'hemi')
l2flow.connect(mergenode, ('out', list2tuple), l2concat, 'vol_measure_file')
```

Perform a one sample t-test

```
l2ttest = pe.Node(interface=fs.OneSampleTTest(), name='onesample')
l2flow.connect(l2concat, 'out_file', l2ttest, 'in_file')
```

Run the analysis pipeline and also create a dot+png (if graphviz is available) that visually represents the workflow.

```
if __name__ == '__main__':  
    l2flow.run()  
    l2flow.write_graph(graph2use='flat')
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

A workflow that uses fsl to perform a first level analysis on the nipype tutorial data set:

```
python fmri_fsl.py
```

First tell python where to find the appropriate functions.

```
from __future__ import print_function
from __future__ import division
from builtins import str
from builtins import range

import os                                # system functions

import nipype.interfaces.io as nio       # Data i/o
import nipype.interfaces.fsl as fsl      # fsl
import nipype.interfaces.utility as util # utility
import nipype.pipeline.engine as pe      # pypeline engine
import nipype.algorithms.modelgen as model # model generation
import nipype.algorithms.rapidart as ra   # artifact detection
```

Preliminaries

Setup any package specific configuration. The output file format for FSL routines is being set to compressed NIFTI.

```
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')
```

Setting up workflows

In this tutorial we will be setting up a hierarchical workflow for fsl analysis. This will demonstrate how pre-defined workflows can be setup and shared across users, projects and labs.

Setup preprocessing workflow

This is a generic fsl feat preprocessing workflow encompassing skull stripping, motion correction and smoothing operations.

```
preproc = pe.Workflow(name='preproc')
```

Set up a node to define all inputs required for the preprocessing workflow

```
inputnode = pe.Node(interface=util.IdentityInterface(fields=['func',
                                                             'struct', ]),
                    name='inputspec')
```

Convert functional images to float representation. Since there can be more than one functional run we use a MapNode to convert each run.

```
img2float = pe.MapNode(interface=fsl.ImageMaths(out_data_type='float',
                                                op_string='',
                                                suffix='_dtype'),
                      iterfield=['in_file'],
                      name='img2float')
preproc.connect(inputnode, 'func', img2float, 'in_file')
```

Extract the middle volume of the first run as the reference

```
extract_ref = pe.Node(interface=fsl.ExtractROI(t_size=1),
                     name='extractref')
```

Define a function to pick the first file from a list of files

```
def pickfirst(files):
    if isinstance(files, list):
        return files[0]
    else:
        return files

preproc.connect(img2float, ('out_file', pickfirst), extract_ref, 'in_file')
```

Define a function to return the 1 based index of the middle volume

```
def getmiddlevolume(func):
    from nibabel import load
    from nipy.utils import NUMPY_MMAP
    funcfile = func
    if isinstance(func, list):
        funcfile = func[0]
    _, _, _, timepoints = load(funcfile, mmap=NUMPY_MMAP).shape
    return int(timepoints / 2) - 1

preproc.connect(inputnode, ('func', getmiddlevolume), extract_ref, 't_min')
```

Realign the functional runs to the middle volume of the first run

```
motion_correct = pe.MapNode(interface=fsl.MCFLIRT(save_mats=True,
                                                  save_plots=True),
                          name='realign',
                          iterfield=['in_file'])
preproc.connect(img2float, 'out_file', motion_correct, 'in_file')
preproc.connect(extract_ref, 'roi_file', motion_correct, 'ref_file')
```

Plot the estimated motion parameters


```

plot_motion = pe.MapNode(interface=fsl.PlotMotionParams(in_source='fsl'),
                          name='plot_motion',
                          iterfield=['in_file'])
plot_motion.iterables = ('plot_type', ['rotations', 'translations'])
preproc.connect(motion_correct, 'par_file', plot_motion, 'in_file')

```

Extract the mean volume of the first functional run

```

meanfunc = pe.Node(interface=fsl.ImageMaths(op_string='-Tmean',
                                             suffix='_mean'),
                   name='meanfunc')
preproc.connect(motion_correct, ('out_file', pickfirst), meanfunc, 'in_file')

```

Strip the skull from the mean functional to generate a mask

```

meanfuncmask = pe.Node(interface=fsl.BET(mask=True,
                                          no_output=True,
                                          frac=0.3),
                       name='meanfuncmask')
preproc.connect(meanfunc, 'out_file', meanfuncmask, 'in_file')

```

Mask the functional runs with the extracted mask

```

maskfunc = pe.MapNode(interface=fsl.ImageMaths(suffix='_bet',
                                                op_string='-mas'),
                      iterfield=['in_file'],
                      name='maskfunc')
preproc.connect(motion_correct, 'out_file', maskfunc, 'in_file')
preproc.connect(meanfuncmask, 'mask_file', maskfunc, 'in_file2')

```

Determine the 2nd and 98th percentile intensities of each functional run

```

getthresh = pe.MapNode(interface=fsl.ImageStats(op_string='-p 2 -p 98'),
                       iterfield=['in_file'],
                       name='getthreshold')
preproc.connect(maskfunc, 'out_file', getthresh, 'in_file')

```

Threshold the first run of the functional data at 10% of the 98th percentile

```

threshold = pe.Node(interface=fsl.ImageMaths(out_data_type='char',
                                             suffix='_thresh'),
                   name='threshold')
preproc.connect(maskfunc, ('out_file', pickfirst), threshold, 'in_file')

```

Define a function to get 10% of the intensity

```

def getthreshop(thresh):
    return '-thr %.10f -Tmin -bin' % (0.1 * thresh[0][1])
preproc.connect(getthresh, ('out_stat', getthreshop), threshold, 'op_string')

```

Determine the median value of the functional runs using the mask

```

medianval = pe.MapNode(interface=fsl.ImageStats(op_string='-k %s -p 50'),
                      iterfield=['in_file'],
                      name='medianval')
preproc.connect(motion_correct, 'out_file', medianval, 'in_file')
preproc.connect(threshold, 'out_file', medianval, 'mask_file')

```

Dilate the mask

```

dilatemask = pe.Node(interface=fsl.ImageMaths(suffix='_dil',
                                              op_string='-dilF'),

```

```

        name='dilatemask')
preproc.connect(threshold, 'out_file', dilatemask, 'in_file')

```

Mask the motion corrected functional runs with the dilated mask

```

maskfunc2 = pe.MapNode(interface=fsl.ImageMaths(suffix='_mask',
                                                op_string='-mas'),
                        iterfield=['in_file'],
                        name='maskfunc2')
preproc.connect(motion_correct, 'out_file', maskfunc2, 'in_file')
preproc.connect(dilatemask, 'out_file', maskfunc2, 'in_file2')

```

Determine the mean image from each functional run

```

meanfunc2 = pe.MapNode(interface=fsl.ImageMaths(op_string='-Tmean',
                                                suffix='_mean'),
                        iterfield=['in_file'],
                        name='meanfunc2')
preproc.connect(maskfunc2, 'out_file', meanfunc2, 'in_file')

```

Merge the median values with the mean functional images into a coupled list

```

mergenode = pe.Node(interface=util.Merge(2, axis='hstack'),
                    name='merge')
preproc.connect(meanfunc2, 'out_file', mergenode, 'in1')
preproc.connect(medianval, 'out_stat', mergenode, 'in2')

```

Smooth each run using SUSAN with the brightness threshold set to 75% of the median value for each run and a mask constituting the mean functional

```

smooth = pe.MapNode(interface=fsl.SUSAN(),
                    iterfield=['in_file', 'brightness_threshold', 'usans'],
                    name='smooth')

```

Define a function to get the brightness threshold for SUSAN

```

def getbtthresh(medianvals):
    return [0.75 * val for val in medianvals]

def getusans(x):
    return [[tuple([val[0], 0.75 * val[1]])] for val in x]

preproc.connect(maskfunc2, 'out_file', smooth, 'in_file')
preproc.connect(medianval, ('out_stat', getbtthresh), smooth, 'brightness_
↳threshold')
preproc.connect(mergenode, ('out', getusans), smooth, 'usans')

```

Mask the smoothed data with the dilated mask

```

maskfunc3 = pe.MapNode(interface=fsl.ImageMaths(suffix='_mask',
                                                op_string='-mas'),
                        iterfield=['in_file'],
                        name='maskfunc3')
preproc.connect(smooth, 'smoothed_file', maskfunc3, 'in_file')
preproc.connect(dilatemask, 'out_file', maskfunc3, 'in_file2')

```

Scale each volume of the run so that the median value of the run is set to 10000

```

intnorm = pe.MapNode(interface=fsl.ImageMaths(suffix='_intnorm'),
                    iterfield=['in_file', 'op_string'],

```

```

        name='intnorm')
preproc.connect(maskfunc3, 'out_file', intnorm, 'in_file')

```

Define a function to get the scaling factor for intensity normalization

```

def getinormscale(medianvals):
    return ['-mul %.10f' % (10000. / val) for val in medianvals]
preproc.connect(medianval, ('out_stat', getinormscale), intnorm, 'op_string')

```

Perform temporal highpass filtering on the data

```

highpass = pe.MapNode(interface=fsl.ImageMaths(suffix='_tempfilt'),
                      iterfield=['in_file'],
                      name='highpass')
preproc.connect(intnorm, 'out_file', highpass, 'in_file')

```

Generate a mean functional image from the first run

```

meanfunc3 = pe.MapNode(interface=fsl.ImageMaths(op_string='-Tmean',
                                                suffix='_mean'),
                      iterfield=['in_file'],
                      name='meanfunc3')
preproc.connect(highpass, ('out_file', pickfirst), meanfunc3, 'in_file')

```

Strip the structural image and coregister the mean functional image to the structural image

```

nosestrip = pe.Node(interface=fsl.BET(frac=0.3),
                    name='nosestrip')
skullstrip = pe.Node(interface=fsl.BET(mask=True),
                    name='stripstruct')

coregister = pe.Node(interface=fsl.FLIRT(dof=6),
                    name='coregister')

```

Use `nipy.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity and/or movement.

```

art = pe.MapNode(interface=ra.ArtifactDetect(use_differences=[True, False],
                                           use_norm=True,
                                           norm_threshold=1,
                                           zintensity_threshold=3,
                                           parameter_source='FSL',
                                           mask_type='file'),
                iterfield=['realigned_files', 'realignment_parameters'],
                name="art")

preproc.connect([(inputnode, nosestrip, [('struct', 'in_file')]),
                (nosestrip, skullstrip, [('out_file', 'in_file')]),
                (skullstrip, coregister, [('out_file', 'in_file')]),
                (meanfunc2, coregister, [(('out_file', pickfirst), 'reference
→ ')]),
                (motion_correct, art, [('par_file', 'realignment_parameters')]),
                (maskfunc2, art, [('out_file', 'realigned_files')]),
                (dilatemask, art, [('out_file', 'mask_file')]),
                ])

```

Set up model fitting workflow

```
modelfit = pe.Workflow(name='modelfit')
```

Use `nipyype.algorithms.modelgen.SpecifyModel` to generate design information.

```
modelspec = pe.Node(interface=model.SpecifyModel(), name="modelspec")
```

Use `nipyype.interfaces.fsl.Level1Design` to generate a run specific fsf file for analysis

```
levelldesign = pe.Node(interface=fsl.Level1Design(), name="levelldesign")
```

Use `nipyype.interfaces.fsl.FEATModel` to generate a run specific mat file for use by FILMGLS

```
modelgen = pe.MapNode(interface=fsl.FEATModel(), name='modelgen',  
                      iterfield=['fsf_file', 'ev_files'])
```

Use `nipyype.interfaces.fsl.FILMGLS` to estimate a model specified by a mat file and a functional run

```
modelestimate = pe.MapNode(interface=fsl.FILMGLS(smooth_autocorr=True,  
                                                mask_size=5,  
                                                threshold=1000),  
                           name='modelestimate',  
                           iterfield=['design_file', 'in_file'])
```

Use `nipyype.interfaces.fsl.ContrastMgr` to generate contrast estimates

```
conestimate = pe.MapNode(interface=fsl.ContrastMgr(), name='conestimate',  
                        iterfield=['tcon_file', 'param_estimates',  
                                  'sigmasquareds', 'corrections',  
                                  'dof_file'])  
  
modelfit.connect([  
    (modelspec, levelldesign, [('session_info', 'session_info')]),  
    (levelldesign, modelgen, [('fsf_files', 'fsf_file'),  
                             ('ev_files', 'ev_files')]),  
    (modelgen, modelestimate, [('design_file', 'design_file')]),  
    (modelgen, conestimate, [('con_file', 'tcon_file')]),  
    (modelestimate, conestimate, [('param_estimates', 'param_estimates'),  
                                  ('sigmasquareds', 'sigmasquareds'),  
                                  ('corrections', 'corrections'),  
                                  ('dof_file', 'dof_file')]),  
])
```

Set up fixed-effects workflow

```
fixed_fx = pe.Workflow(name='fixedfx')
```

Use `nipyype.interfaces.fsl.Merge` to merge the copes and varcopes for each condition

```
copemerge = pe.MapNode(interface=fsl.Merge(dimension='t'),  
                      iterfield=['in_files'],  
                      name="copemerge")  
  
varcopemerge = pe.MapNode(interface=fsl.Merge(dimension='t'),  
                          iterfield=['in_files'],  
                          name="varcopemerge")
```

Use `nipype.interfaces.fsl.L2Model` to generate subject and condition specific level 2 model design files

```
level2model = pe.Node(interface=fsl.L2Model(),
                      name='l2model')
```

Use `nipype.interfaces.fsl.FLAMEO` to estimate a second level model

```
flameo = pe.MapNode(interface=fsl.FLAMEO(run_mode='fe'), name="flameo",
                    iterfield=['cope_file', 'var_cope_file'])

fixed_fx.connect([(copemerge, flameo, [('merged_file', 'cope_file')]),
                  (varcopemerge, flameo, [('merged_file', 'var_cope_file')]),
                  (level2model, flameo, [('design_mat', 'design_file'),
                                          ('design_con', 't_con_file'),
                                          ('design_grp', 'cov_split_file')]),
                  ])
```

Set up first-level workflow

```
def sort_copes(files):
    numelements = len(files[0])
    outfiles = []
    for i in range(numelements):
        outfiles.insert(i, [])
        for j, elements in enumerate(files):
            outfiles[i].append(elements[i])
    return outfiles

def num_copes(files):
    return len(files)

firstlevel = pe.Workflow(name='firstlevel')
firstlevel.connect([(preproc, modelfit, [('highpass.out_file', 'modelspec.
↳functional_runs'),
                                          ('art.outlier_files', 'modelspec.outlier_
↳files'),
                                          ('highpass.out_file', 'modelestimate.in_
↳file')]),
                    (preproc, fixed_fx, [('coregister.out_file', 'flameo.mask_file
↳')]),
                    (modelfit, fixed_fx, [(['conestimate.copes', sort_copes),
↳'copemerge.in_files'],
                                          [(['conestimate.varcopies', sort_copes),
↳'varcopemerge.in_files'],
                                          [(['conestimate.copes', num_copes),
↳'l2model.num_copes'],
                                          ]))])
```

Experiment specific components

The nipype tutorial contains data for two subjects. Subject data is in two subdirectories, `s1` and `s2`. Each subject directory contains four functional volumes: `f3.nii`, `f5.nii`, `f7.nii`, `f10.nii`. And one anatomical volume

named `struct.nii`.

Below we set some variables to inform the `datasource` about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (`struct` or `func`). These fields become the output fields of the `datasource` node in the pipeline.

In the example below, run 'f3' is of type 'func' and gets mapped to a nifti filename through a template '%s.nii'. So 'f3' would become 'f3.nii'.

```
# Specify the location of the data.
data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1'] # , 's3'
# Map field names to individual subject runs.
info = dict(func=['subject_id', ['f3', 'f5', 'f7', 'f10']],
            struct=['subject_id', 'struct'])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                    name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a `nipyype.interfaces.io.DataSource` object and fill in the information from above about the layout of our data. The `nipyype.pipeline.NodeWrapper` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                             outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Use the `get_node` function to retrieve an internal node by name. Then set the iterables on this node to perform two different extents of smoothing.

```
smoothnode = firstlevel.get_node('preproc.smooth')
assert(str(smoothnode) == 'preproc.smooth')
smoothnode.iterables = ('fwhm', [5., 10.])

hpcutoff = 120
TR = 3. # ensure float
firstlevel.inputs.preproc.highpass.suffix = '_hpf'
firstlevel.inputs.preproc.highpass.op_string = '-bptf %d -1' % (hpcutoff / TR)
```

Setup a function that returns subject-specific information about the experimental paradigm. This is used by the `nipyype.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant. Other examples of this function are available in the `doc/examples` folder. Note: Python knowledge required here.

```
def subjectinfo(subject_id):
    from nipyype.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: %s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
```

```

for r in range(4):
    onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
    output.insert(r,
                  Bunch(conditions=names,
                        onsets=deepcopy(onsets),
                        durations=[[15] for s in names],
                        amplitudes=None,
                        tmod=None,
                        pmod=None,
                        regressor_names=None,
                        regressors=None))

return output

```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]]. The condition names must match the *names* listed in the *subjectinfo* function described above.

```

cont1 = ['Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5]]
cont2 = ['Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1]]
cont3 = ['Task', 'F', [cont1, cont2]]
contrasts = [cont1, cont2]

firstlevel.inputs.modelfit.modelspec.input_units = 'secs'
firstlevel.inputs.modelfit.modelspec.time_repetition = TR
firstlevel.inputs.modelfit.modelspec.high_pass_filter_cutoff = hpcutoff

firstlevel.inputs.modelfit.level1design.interscan_interval = TR
firstlevel.inputs.modelfit.level1design.bases = {'dgamma': {'derivs': False}}
firstlevel.inputs.modelfit.level1design.contrasts = contrasts
firstlevel.inputs.modelfit.level1design.model_serial_correlations = True

```

Set up complete workflow

```

llpipeline = pe.Workflow(name="level1")
llpipeline.base_dir = os.path.abspath('./fsl/workingdir')
llpipeline.config = {"execution": {"crashdump_dir": os.path.abspath('./fsl/
↳crashdumps')}}

llpipeline.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                    (infosource, firstlevel, [('subject_id', subjectinfo),
↳'modelfit.modelspec.subject_info'])),
                    (datasource, firstlevel, [('struct', 'preproc.inputs.spec.struct
↳'),
                                                ('func', 'preproc.inputs.spec.func'),
                                                ]),
                    ])

```

Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipyype.pipeline.engine.Pipeline.Run` function needs to be called.

```

if __name__ == '__main__':
    llpipeline.write_graph()

```

```
outgraph = llpipeline.run()
# llpipeline.run(plugin='MultiProc', plugin_args={'n_procs':2})
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

fMRI: FEEDS - FSL

A pipeline example that data from the FSL FEEDS set. Single subject, two stimuli.
 You can find it at <http://www.fmrib.ox.ac.uk/fsl/feeds/doc/index.html>

```
from __future__ import division
from builtins import range

import os
from nipy.interfaces import io as nio
from nipy.interfaces import utility as niu
from nipy.interfaces import fsl
from nipy.pipeline import engine as pe
from nipy.algorithms import modelgen as model
from nipy.workflows.fmri.fsl import (create_featreg_preproc,
                                     create_modelfit_workflow,
                                     create_reg_workflow)
from nipy.interfaces.base import Bunch
```

Preliminaries

Setup any package specific configuration. The output file format for FSL routines is being set to compressed NIFTI.

```
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')
```

Experiment specific components

This tutorial does a single subject analysis so we are not using inforsource and iterables

```
# Specify the location of the FEEDS data. You can find it at http://www.fmrib.ox.
↪ac.uk/fsl/feeds/doc/index.html

inputnode = pe.Node(niu.IdentityInterface(fields=['in_data']), name='inputnode')
# Specify the subject directories
# Map field names to individual subject runs.
```

```
info = dict(func=['fmri'],
            struct=['structural'])
```

Now we create a `nipyne.interfaces.io.DataSource` object and fill in the information from above about the layout of our data. The `nipyne.pipeline.Node` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.template = 'feeds/data/%s.nii.gz'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

preproc = create_featreg_preproc(whichvol='first')
TR = 3.
preproc.inputs.inputspec.fwhm = 5
preproc.inputs.inputspec.highpass = 100. / TR

modelspec = pe.Node(interface=model.SpecifyModel(),
                    name="modelspec")
modelspec.inputs.input_units = 'secs'
modelspec.inputs.time_repetition = TR
modelspec.inputs.high_pass_filter_cutoff = 100
modelspec.inputs.subject_info = [Bunch(conditions=['Visual', 'Auditory'],
                                       onsets=[list(range(0, int(180 * TR), 60)),
↪list(range(0, int(180 * TR), 90))],
                                       durations=[[30], [45]],
                                       amplitudes=None,
                                       tmod=None,
                                       pmod=None,
                                       regressor_names=None,
                                       regressors=None)]

modelfit = create_modelfit_workflow(f_contrasts=True)
modelfit.inputs.inputspec.interscan_interval = TR
modelfit.inputs.inputspec.model_serial_correlations = True
modelfit.inputs.inputspec.bases = {'dgamma': {'derivs': True}}
cont1 = ['Visual>Baseline', 'T', ['Visual', 'Auditory'], [1, 0]]
cont2 = ['Auditory>Baseline', 'T', ['Visual', 'Auditory'], [0, 1]]
cont3 = ['Task', 'F', [cont1, cont2]]
modelfit.inputs.inputspec.contrasts = [cont1, cont2, cont3]

registration = create_reg_workflow()
registration.inputs.inputspec.target_image = fsl.Info.standard_image('MNI152_T1_
↪2mm.nii.gz')
registration.inputs.inputspec.target_image_brain = fsl.Info.standard_image(
↪'MNI152_T1_2mm_brain.nii.gz')
registration.inputs.inputspec.config_file = 'T1_2_MNI152_2mm'
```

Set up complete workflow

```
l1pipeline = pe.Workflow(name="level1")
l1pipeline.base_dir = os.path.abspath('./fsl_feeds/workingdir')
l1pipeline.config = {"execution": {"crashdump_dir": os.path.abspath('./fsl_feeds/
↪crashdumps')}}

l1pipeline.connect(inputnode, 'in_data', datasource, 'base_directory')
```

```

llpipeline.connect(datasource, 'func', preproc, 'inputspec.func')
llpipeline.connect(preproc, 'outputspec.highpassed_files', modelspec, 'functional_
↳runs')
llpipeline.connect(preproc, 'outputspec.motion_parameters', modelspec,
↳'realignment_parameters')
llpipeline.connect(modelspec, 'session_info', modelfit, 'inputspec.session_info')
llpipeline.connect(preproc, 'outputspec.highpassed_files', modelfit, 'inputspec.
↳functional_data')
llpipeline.connect(preproc, 'outputspec.mean', registration, 'inputspec.mean_image
↳')
llpipeline.connect(datasource, 'struct', registration, 'inputspec.anatomical_image
↳')
llpipeline.connect(modelfit, 'outputspec.zfiles', registration, 'inputspec.source_
↳files')

```

Setup the datasink

```

datasink = pe.Node(interface=nio.DataSink(parameterization=False), name="datasink
↳")
datasink.inputs.base_directory = os.path.abspath('./fsl_feeds/llout')
datasink.inputs.substitutions = [('fmri_dtype_mcf_mask_smooth_mask_gms_mean_warp',
↳ 'meanfunc')]
# store relevant outputs from various stages of the 1st level analysis
llpipeline.connect(registration, 'outputspec.transformed_files', datasink,
↳'level1.@Z')
llpipeline.connect(registration, 'outputspec.transformed_mean', datasink,
↳'meanfunc')

```

Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipy.pipeline.engine.Pipeline.Run` function needs to be called.

```

if __name__ == '__main__':
    lllpipeline.inputs.inputnode.in_data = os.path.abspath('feeds/data')
    lllpipeline.run()

```

Example source code

You can download the full source code of this example. This same script is also included in the Nipy source distribution under the `examples` directory.

fMRI: FSL reuse workflows

A workflow that uses fsl to perform a first level analysis on the nipype tutorial data set:

```
python fmri_fsl_reuse.py
```

First tell python where to find the appropriate functions.

```
from __future__ import print_function
from __future__ import division
from builtins import str
from builtins import range

import os                                # system functions
import nipype.interfaces.io as nio       # Data i/o
import nipype.interfaces.fsl as fsl     # fsl
from nipype.interfaces import utility as niu # Utilities
import nipype.pipeline.engine as pe      # pypeline engine
import nipype.algorithms.modelgen as model # model generation
import nipype.algorithms.rapidart as ra   # artifact detection

from nipype.workflows.fmri.fsl import (create_featreg_preproc,
                                       create_modelfit_workflow,
                                       create_fixed_effects_flow)
```

Preliminaries

Setup any package specific configuration. The output file format for FSL routines is being set to compressed NIFTI.

```
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')

levell_workflow = pe.Workflow(name='levellflow')

preproc = create_featreg_preproc(whichvol='first')

modelfit = create_modelfit_workflow()

fixed_fx = create_fixed_effects_flow()
```

Add artifact detection and model specification nodes between the preprocessing and modelfitting workflows.

```
art = pe.MapNode(ra.ArtifactDetect(use_differences=[True, False],
                                   use_norm=True,
                                   norm_threshold=1,
                                   zintensity_threshold=3,
                                   parameter_source='FSL',
                                   mask_type='file'),
                 iterfield=['realigned_files', 'realignment_parameters', 'mask_
→file'],
                 name="art")

modelspec = pe.Node(model.SpecifyModel(), name="modelspec")

levell_workflow.connect([(preproc, art, [('outputspec.motion_parameters',
                                         'realignment_parameters'),
                                         ('outputspec.realigned_files',
                                         'realigned_files'),
                                         ('outputspec.mask', 'mask_file')]),
                        (preproc, modelspec, [('outputspec.highpassed_files',
                                         'functional_runs'),
                                         ('outputspec.motion_parameters',
                                         'realignment_parameters')]),
                        (art, modelspec, [('outlier_files', 'outlier_files')]),
                        (modelspec, modelfit, [('session_info', 'inputspec.
→session_info')]),
                        (preproc, modelfit, [('outputspec.highpassed_files',
→'inputspec.functional_data')])
                        ])
```

Set up first-level workflow

```
def sort_copes(files):
    numelements = len(files[0])
    outfiles = []
    for i in range(numelements):
        outfiles.insert(i, [])
        for j, elements in enumerate(files):
            outfiles[i].append(elements[i])
    return outfiles

def num_copes(files):
    return len(files)

pickfirst = lambda x: x[0]

levell_workflow.connect([(preproc, fixed_fx, [('outputspec.mask', pickfirst),
                                         'flameo.mask_file']]),
                        (modelfit, fixed_fx, [('outputspec.copes', sort_copes),
                                         'inputspec.copes'),
                                         ('outputspec.dof_file',
                                         'inputspec.dof_files'),
                                         ('outputspec.varcopes',
                                         sort_copes),
```

```

        'inputspec.varcopes'),
        (('outputspec.copes', num_copes),
         'l2model.num_copes'),
    ])
1)

```

Experiment specific components

The nipy tutorial contains data for two subjects. Subject data is in two subdirectories, `s1` and `s2`. Each subject directory contains four functional volumes: `f3.nii`, `f5.nii`, `f7.nii`, `f10.nii`. And one anatomical volume named `struct.nii`.

Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (`struct` or `func`). These fields become the output fields of the datasource node in the pipeline. In the example below, run 'f3' is of type 'func' and gets mapped to a nifti filename through a template '%s.nii'. So 'f3' would become 'f3.nii'.

```

inputnode = pe.Node(niu.IdentityInterface(fields=['in_data']), name='inputnode')

# Specify the subject directories
subject_list = ['s1'] # , 's3']
# Map field names to individual subject runs.
info = dict(func=[['subject_id', ['f3', 'f5', 'f7', 'f10']]],
            struct=[['subject_id', 'struct']])

infosource = pe.Node(niu.IdentityInterface(fields=['subject_id']),
                    name="infosource")

```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```

infosource.iterables = ('subject_id', subject_list)

```

Now we create a `nipy.interfaces.io.DataSource` object and fill in the information from above about the layout of our data. The `nipy.pipeline.NodeWrapper` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```

datasource = pe.Node(nio.DataGrabber(infields=['subject_id'],
                                     outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.template = 'nipy-tutorial/data/%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

```

Use the `get_node` function to retrieve an internal node by name. Then set the iterables on this node to perform two different extents of smoothing.

```

featininput = level1_workflow.get_node('featpreproc.inputspec')
featininput.iterables = ('fwhm', [5., 10.])

hpcutoff = 120.
TR = 3.
featininput.inputs.highpass = hpcutoff / (2. * TR)

```

Setup a function that returns subject-specific information about the experimental paradigm. This is used by the `nipy.modelgen.SpecifyModel` to create the information necessary to generate an SPM design

matrix. In this tutorial, the same paradigm was used for every participant. Other examples of this function are available in the *doc/examples* folder. Note: Python knowledge required here.

```
def subjectinfo(subject_id):
    from nipyype.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: %s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r,
                      Bunch(conditions=names,
                            onsets=deepcopy(onsets),
                            durations=[[15] for s in names]))
    return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]]. The condition names must match the *names* listed in the *subjectinfo* function described above.

```
cont1 = ['Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5]]
cont2 = ['Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1]]
cont3 = ['Task', 'F', [cont1, cont2]]
contrasts = [cont1, cont2]

modelspec.inputs.input_units = 'secs'
modelspec.inputs.time_repetition = TR
modelspec.inputs.high_pass_filter_cutoff = hpcutoff

modelfit.inputs.inputs.spec.interscan_interval = TR
modelfit.inputs.inputs.spec.bases = {'dgamma': {'derivs': False}}
modelfit.inputs.inputs.spec.contrasts = contrasts
modelfit.inputs.inputs.spec.model_serial_correlations = True
modelfit.inputs.inputs.spec.film_threshold = 1000

levell_workflow.base_dir = os.path.abspath('./fsl/workingdir')
levell_workflow.config['execution'] = dict(crashdump_dir=os.path.abspath('./fsl/
→crashdumps'))

levell_workflow.connect([(inputnode, datasource, [('in_data', 'base_directory')]),
                        (infosource, datasource, [('subject_id', 'subject_id')]),
                        (infosource, modelspec, [('subject_id', subjectinfo),
                                                  'subject_info'])),
                        (datasource, preproc, [('func', 'inputs.spec.func')]),
                        ])
```

Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipyype.pipeline.engine.Pipeline.Run` function needs to be called.

```
if __name__ == '__main__':
    # levell_workflow.write_graph()
    levell_workflow.run()
    # levell_workflow.run(plugin='MultiProc', plugin_args={'n_procs':2})
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

fMRI: NiPy GLM, SPM

The `fmri_nipy_glm.py` integrates several interfaces to perform a first level analysis on a two-subject data set. It is very similar to the `spm_tutorial` with the difference of using `nipy` for fitting GLM model and estimating contrasts. The tutorial can be found in the examples folder. Run the tutorial from inside the `nipy` tutorial directory:

```
python fmri_nipy_glm.py
```

```
from __future__ import print_function
from builtins import str
from builtins import range

from nipy.interfaces.nipy.model import FitGLM, EstimateContrast
from nipy.interfaces.nipy.preprocess import ComputeMask
```

Import necessary modules from `nipy`.

```
import nipy.interfaces.io as nio           # Data i/o
import nipy.interfaces.spm as spm         # spm
import nipy.interfaces.matlab as mlab     # how to run matlab
import nipy.interfaces.utility as util    # utility
import nipy.pipeline.engine as pe        # pypeline engine
import nipy.algorithms.rapidart as ra     # artifact detection
import nipy.algorithms.modelgen as model # model specification
import os                                 # system functions
```

Preliminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does not handle compressed NIFTI.

```
# Set the way matlab should be called
mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
```

The `nipy` tutorial contains data for two subjects. Subject data is in two subdirectories, `s1` and `s2`. Each subject directory contains four functional volumes: `f3.nii`, `f5.nii`, `f7.nii`, `f10.nii`. And one anatomical volume

named `struct.nii`.

Below we set some variables to inform the `datasource` about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (`struct` or `func`). These fields become the output fields of the `datasource` node in the pipeline.

In the example below, run 'f3' is of type 'func' and gets mapped to a nifti filename through a template '%s.nii'. So 'f3' would become 'f3.nii'.

```
# Specify the location of the data.
data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1']
# Map field names to individual subject runs.
info = dict(func=['subject_id', ['f3', 'f5', 'f7', 'f10']],
            struct=['subject_id', 'struct'])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                    name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```
infosource.iterables = ('subject_id', subject_list)
```

Preprocessing pipeline nodes

Now we create a `nipyype.interfaces.io.DataSource` object and fill in the information from above about the layout of our data. The `nipyype.pipeline.NodeWrapper` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                              outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Use `nipyype.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```
realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True

compute_mask = pe.Node(interface=ComputeMask(), name="compute_mask")
```

Use `nipyype.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```
art = pe.Node(interface=ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```

Use `nipyype.interfaces.spm.Coregister` to perform a rigid body registration of the functional data to the structural data.

```
coregister = pe.Node(interface=spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
```

Smooth the functional data using `nipyype.interfaces.spm.Smooth`.

```
smooth = pe.Node(interface=spm.Smooth(), name="smooth")
smooth.inputs.fwhm = 4
```

Set up analysis components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the `nipyype.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant.

```
def subjectinfo(subject_id):
    from nipyype.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: %s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r,
                      Bunch(conditions=names,
                            onsets=deepcopy(onsets),
                            durations=[[15] for s in names],
                            amplitudes=None,
                            tmod=None,
                            pmod=None,
                            regressor_names=None,
                            regressors=None))
    return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name, Stat, [list of condition names], [weights on those conditions]]. The condition names must match the *names* listed in the *subjectinfo* function described above.

```
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]
```

Generate design information using `nipyype.interfaces.spm.SpecifyModel`. `nipy` accepts only design specified in seconds so “output_units” has always have to be set to “secs”.

```
modelspec = pe.Node(interface=model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = True
modelspec.inputs.input_units = 'secs'
modelspec.inputs.output_units = 'secs'
modelspec.inputs.time_repetition = 3.
modelspec.inputs.high_pass_filter_cutoff = 120
```

Fit the GLM model using `nipy` and ordinary least square method

```
model_estimate = pe.Node(interface=FitGLM(), name="model_estimate")
model_estimate.inputs.TR = 3.
model_estimate.inputs.model = "spherical"
model_estimate.inputs.method = "ols"
```

Estimate the contrasts. The format of the contrasts definition is the same as for FSL and SPM

```

contrast_estimate = pe.Node(interface=EstimateContrast(), name="contrast_estimate
↪")
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrast_estimate.inputs.contrasts = [cont1, cont2]

```

Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the `nipy.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use *workdir* as the disk location to use when running the processes and keeping their outputs. The *use_parameterized_dirs* tells the engine to create sub-directories under *workdir* corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The `nipy.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

```

l1pipeline = pe.Workflow(name="level1")
l1pipeline.base_dir = os.path.abspath('nipy_tutorial/workingdir')

l1pipeline.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                    (datasource, realign, [('func', 'in_files')]),
                    (realign, compute_mask, [('mean_image', 'mean_volume')]),
                    (realign, coregister, [('mean_image', 'source'),
↪                                     ('realigned_files', 'apply_to_files
↪')]),
                    (datasource, coregister, [('struct', 'target')]),
                    (coregister, smooth, [('coregistered_files', 'in_files')]),
                    (realign, modelspec, [('realignment_parameters', 'realignment_
↪parameters')]),
                    (smooth, modelspec, [('smoothed_files', 'functional_runs')]),
                    (realign, art, [('realignment_parameters', 'realignment_
↪parameters')]),
                    (coregister, art, [('coregistered_files', 'realigned_files
↪')]),
                    (compute_mask, art, [('brain_mask', 'mask_file')]),
                    (art, modelspec, [('outlier_files', 'outlier_files')]),
                    (infosource, modelspec, [('"subject_id", subjectinfo),
↪"subject_info"]]),
                    (modelspec, model_estimate, [('session_info', 'session_info
↪')]),
                    (compute_mask, model_estimate, [('brain_mask', 'mask')]),
                    (model_estimate, contrast_estimate, [('"beta", "beta"),
                                                         ("nvbeta", "nvbeta"),
                                                         ("s2", "s2"),
                                                         ("dof", "dof"),
                                                         ("axis", "axis"),
                                                         ("constants", "constants
↪"),
                                                         ("reg_names", "reg_names
↪")])])

if __name__ == '__main__':

```

```
llpipeline.run()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipyype source distribution under the `examples` directory.

Example source code

You can download the full source code of this example. This same script is also included in the Nipyype source distribution under the `examples` directory.

fmRI: Coregistration - Slicer, BRAINS

This is currently not working and will raise an exception in release 0.3. It will be fixed in a later release:

```
python fmri_slicer_coregistration.py
```

```
# raise RuntimeError, 'Slicer not fully implemented'
from nipy.interfaces.slicer import BRAINSFit, BRAINSResample
```

Import necessary modules from nipy.

```
import nipy.interfaces.io as nio          # Data i/o
import nipy.interfaces.utility as util    # utility
import nipy.pipeline.engine as pe        # pipeline engine
import os                                 # system functions
```

Preliminaries

Confirm package dependencies are installed. (This is only for the tutorial, rarely would you put this in your own code.)

```
from nipy.utils.misc import package_check

package_check('numpy', '1.3', 'tutorial1')
package_check('scipy', '0.7', 'tutorial1')
package_check('networkx', '1.0', 'tutorial1')
package_check('IPython', '0.10', 'tutorial1')
```

The nipy tutorial contains data for two subjects. Subject data is in two subdirectories, `s1` and `s2`. Each subject directory contains four functional volumes: `f3.nii`, `f5.nii`, `f7.nii`, `f10.nii`. And one anatomical volume named `struct.nii`.

Below we set some variables to inform the `datasource` about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (`struct` or `func`). These fields become the output fields of the `datasource` node in the pipeline.

In the example below, run `'f3'` is of type `'func'` and gets mapped to a nifti filename through a template `'%s.nii'`. So `'f3'` would become `'f3.nii'`.

```
# Specify the location of the data.
data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1', 's3']
# Map field names to individual subject runs.
info = dict(func=['subject_id', 'f3'],
            struct=['subject_id', 'struct'])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                    name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```
infosource.iterables = ('subject_id', subject_list)
```

Preprocessing pipeline nodes

Now we create a `nipy.interfaces.io.DataSource` object and fill in the information from above about the layout of our data. The `nipy.pipeline.NodeWrapper` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                              outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

coregister = pe.Node(interface=BRAINSFit(), name="coregister")
coregister.inputs.outputTransform = True
coregister.inputs.outputVolume = True
coregister.inputs.transformType = ["Affine"]

reslice = pe.Node(interface=BRAINSResample(), name="reslice")
reslice.inputs.outputVolume = True

pipeline = pe.Workflow(name="pipeline")
pipeline.base_dir = os.path.abspath('slicer_tutorial/workingdir')

pipeline.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                 (datasource, coregister, [('func', 'movingVolume')]),
                 (datasource, coregister, [('struct', 'fixedVolume')]),
                 (coregister, reslice, [('outputTransform', 'warpTransform')]),
                 (datasource, reslice, [('func', 'inputVolume')]),
                 (datasource, reslice, [('struct', 'referenceVolume'])
                 ]))

if __name__ == '__main__':
    pipeline.run()
    pipeline.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

CHAPTER 29

fMRI: SPM, FSL

The `fmri_spm.py` integrates several interfaces to perform a first and second level analysis on a two-subject data set. The tutorial can be found in the examples folder. Run the tutorial from inside the nipype tutorial directory:

```
python fmri_spm.py
```

Import necessary modules from nipype.

```
from __future__ import print_function
from builtins import str
from builtins import range

import os                                     # system functions

from nipype import config
# config.enable_provenance()

from nipype.interfaces import spm, fsl

# In order to use this example with SPM's matlab common runtime
# matlab_cmd = ('/Users/satra/Downloads/spm8/run_spm8.sh '
#              '/Applications/MATLAB/MATLAB_Compiler_Runtime/v713/ script')
# spm.SPMCommand.set_mlab_paths(matlab_cmd=matlab_cmd, use_mcr=True)

import nipype.interfaces.io as nio           # Data i/o
import nipype.interfaces.utility as util     # utility
import nipype.pipeline.engine as pe         # pypeline engine
import nipype.algorithms.rapidart as ra     # artifact detection
import nipype.algorithms.modelgen as model  # model specification
import nipype.interfaces.matlab as mlab
```

Preliminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does not handle compressed NIFTI.

```
# Tell fsl to generate all output in uncompressed nifti format
fsl.FSLCommand.set_default_output_type('NIFTI')

# Set the way matlab should be called
# import nipy.interfaces.matlab as mlab          # how to run matlab
# mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")

# In case a different path is required
# mlab.MatlabCommand.set_default_paths('/software/matlab/spm12b/spm12b_r5918')
```

The nipy tutorial contains data for two subjects. Subject data is in two subdirectories, `s1` and `s2`. Each subject directory contains four functional volumes: `f3.nii`, `f5.nii`, `f7.nii`, `f10.nii`. And one anatomical volume named `struct.nii`.

Below we set some variables to inform the `datasource` about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (`struct` or `func`). These fields become the output fields of the `datasource` node in the pipeline. In the example below, run `'f3'` is of type `'func'` and gets mapped to a nifti filename through a template `'%s.nii'`. So `'f3'` would become `'f3.nii'`.

```
# Specify the location of the data.
data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1', 's3']
# Map field names to individual subject runs.
info = dict(func=['subject_id', ['f3', 'f5', 'f7', 'f10']],
            struct=['subject_id', 'struct'])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                    name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The `datasource` attribute `iterables` tells the pipeline engine that it should repeat the analysis on each of the items in the `subject_list`. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in `subject_list`.

```
infosource.iterables = ('subject_id', subject_list)
```

Preprocessing pipeline nodes

Now we create a `nipy.interfaces.io.DataSource` object and fill in the information from above about the layout of our data. The `nipy.pipeline.NodeWrapper` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                                outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Use `nipy.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```
realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True
```

Use `nipy.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```
art = pe.Node(interface=ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```

Skull strip structural images using `nipype.interfaces.fsl.BET`.

```
skullstrip = pe.Node(interface=fsl.BET(), name="skullstrip")
skullstrip.inputs.mask = True
```

Use `nipype.interfaces.spm.Coregister` to perform a rigid body registration of the functional data to the structural data.

```
coregister = pe.Node(interface=spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
```

Warp functional and structural data to SPM's T1 template using `nipype.interfaces.spm.Normalize`. The tutorial data set includes the template image, T1.nii.

```
normalize = pe.Node(interface=spm.Normalize(), name="normalize")
normalize.inputs.template = os.path.abspath('data/T1.nii')
```

Smooth the functional data using `nipype.interfaces.spm.Smooth`.

```
smooth = pe.Node(interface=spm.Smooth(), name="smooth")
fwhmlist = [4]
smooth.iterables = ('fwhm', fwhmlist)
```

Set up analysis components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the `nipype.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant.

```
def subjectinfo(subject_id):
    from nipype.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: %s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r,
                      Bunch(conditions=names,
                           onsets=deepcopy(onsets),
                           durations=[[15] for s in names]))
    return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name, Stat, [list of condition names], [weights on those conditions]]. The condition names must match the *names* listed in the *subjectinfo* function described above.

```
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]
```

Generate SPM-specific design information using `nipy.interfaces.spm.SpecifyModel`.

```
modelspec = pe.Node(interface=model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = False
modelspec.inputs.input_units = 'secs'
modelspec.inputs.output_units = 'secs'
modelspec.inputs.time_repetition = 3.
modelspec.inputs.high_pass_filter_cutoff = 120
```

Generate a first level SPM.mat file for analysis `nipy.interfaces.spm.Level1Design`.

```
levelldesign = pe.Node(interface=spm.Level1Design(), name="levelldesign")
levelldesign.inputs.timing_units = modelspec.inputs.output_units
levelldesign.inputs.interscan_interval = modelspec.inputs.time_repetition
levelldesign.inputs.bases = {'hrf': {'derivs': [0, 0]}}
```

Use `nipy.interfaces.spm.EstimateModel` to determine the parameters of the model.

```
levelleestimate = pe.Node(interface=spm.EstimateModel(), name="levelleestimate")
levelleestimate.inputs.estimate_method = {'Classical': 1}
```

Use `nipy.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.

```
contrastestimate = pe.Node(interface=spm.EstimateContrast(), name=
    → "contrastestimate")
contrastestimate.inputs.contrasts = contrasts
contrastestimate.overwrite = True
contrastestimate.config = {'execution': {'remove_unnecessary_outputs': False}}
```

Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the `nipy.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use *workdir* as the disk location to use when running the processes and keeping their outputs. The *use_parameterized_dirs* tells the engine to create sub-directories under *workdir* corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The `nipy.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

```
l1pipeline = pe.Workflow(name="level1")
l1pipeline.base_dir = os.path.abspath('spm_tutorial/workingdir')

l1pipeline.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                    (datasource, realign, [('func', 'in_files')]),
                    (realign, coregister, [('mean_image', 'source'),
                                             ('realigned_files', 'apply_to_files'
→ ')]),
                    (datasource, coregister, [('struct', 'target')]),
                    (datasource, normalize, [('struct', 'source')]),
                    (coregister, normalize, [('coregistered_files', 'apply_to_
→ files')]),
                    (normalize, smooth, [('normalized_files', 'in_files')]),
                    (infosource, modelspec, [(('subject_id', subjectinfo),
                                              'subject_info')])])
```



```

        (realign, modelspec, [('realignment_parameters', 'realignment_
↪parameters')])),
        (smooth, modelspec, [('smoothed_files', 'functional_runs')])),
        (normalize, skullstrip, [('normalized_source', 'in_file')])),
        (realign, art, [('realignment_parameters', 'realignment_
↪parameters')])),
        (normalize, art, [('normalized_files', 'realigned_files')])),
        (skullstrip, art, [('mask_file', 'mask_file')])),
        (art, modelspec, [('outlier_files', 'outlier_files')])),
        (modelspec, levelldesign, [('session_info', 'session_info')])),
        (skullstrip, levelldesign, [('mask_file', 'mask_image')])),
        (levelldesign, levelleestimate, [('spm_mat_file', 'spm_mat_file
↪')])),
        (levelleestimate, contrastestimate, [('spm_mat_file', 'spm_mat_
↪file'),
                                                    ('beta_images', 'beta_
↪images'),
                                                    ('residual_image',
↪'residual_image')])),
    ])

```

Setup storage results

Use `nipyype.interfaces.io.DataSink` to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them.

The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form `directory_name[@subdir]` where parts between `[]` are optional. For example `'realign.@mean'` below creates a directory called `realign` in `'11output/subject_id/'` and stores the mean image output from the Realign process in the `realign` directory. If the `@` is left out, then a sub-directory with the name `'mean'` would be created and the mean image would be copied to that directory.

```

datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('spm_tutorial/11output')

def getstripdir(subject_id):
    import os
    return os.path.join(os.path.abspath('spm_tutorial/workingdir'), '_subject_id_
↪%s' % subject_id)

# store relevant outputs from various stages of the 1st level analysis
l1pipeline.connect([(infosource, datasink, [('subject_id', 'container'),
                                                    (('subject_id', getstripdir), 'strip_
↪dir')])),
                    (realign, datasink, [('mean_image', 'realign.@mean'),
                                                    ('realignment_parameters', 'realign.
↪@param')])),
                    (art, datasink, [('outlier_files', 'art.@outliers'),
                                                    ('statistic_files', 'art.@stats')])),
                    (levelldesign, datasink, [('spm_mat_file', 'model.pre-estimate
↪')])),
                    (levelleestimate, datasink, [('spm_mat_file', 'model.@spm'),
                                                    ('beta_images', 'model.@beta'),
                                                    ('mask_image', 'model.@mask'),
                                                    ('residual_image', 'model.@res'),
                                                    ('RPVimage', 'model.@rpv')])),

```

```

        (contrastestimate, datasink, [('con_images', 'contrasts.@con
↪'),
                                     ('spmT_images', 'contrasts.@T
↪')]),
    ])

```

Setup level 2 pipeline

Use `nipyre.interfaces.io.DataGrabber` to extract the contrast images across a group of first level subjects. Unlike the previous pipeline that iterated over subjects, this pipeline will iterate over contrasts.

```

# collect all the con images for each contrast.
contrast_ids = list(range(1, len(contrasts) + 1))
l2source = pe.Node(nio.DataGrabber(infields=['fwhm', 'con']), name="l2source")
# we use .*i* to capture both .img (SPM8) and .nii (SPM12)
l2source.inputs.template = os.path.abspath('spm_tutorial/l1output/*/con*/*_fwhm_
↪%d/con_%04d.*i*')
# iterate over all contrast images
l2source.iterables = [('fwhm', fwhmlist),
                     ('con', contrast_ids)]
l2source.inputs.sort_filelist = True

```

Use `nipyre.interfaces.spm.OneSampleTTestDesign` to perform a simple statistical analysis of the contrasts from the group of subjects ($n=2$ in this example).

```

# setup a 1-sample t-test node
onesampttestdes = pe.Node(interface=spm.OneSampleTTestDesign(), name=
↪"onesampttestdes")
l2estimate = pe.Node(interface=spm.EstimateModel(), name="level2estimate")
l2estimate.inputs.estimate_method = {'Classical': 1}
l2conestimate = pe.Node(interface=spm.EstimateContrast(), name="level2conestimate
↪")
cont1 = ('Group', 'T', ['mean'], [1])
l2conestimate.inputs.contrasts = [cont1]
l2conestimate.inputs.group_contrast = True

```

As before, we setup a pipeline to connect these two nodes (`l2source -> onesampttest`).

```

l2pipeline = pe.Workflow(name="level2")
l2pipeline.base_dir = os.path.abspath('spm_tutorial/l2output')
l2pipeline.connect([(l2source, onesampttestdes, [('outfiles', 'in_files')]),
                   (onesampttestdes, l2estimate, [('spm_mat_file', 'spm_mat_
↪file')]),
                   (l2estimate, l2conestimate, [('spm_mat_file', 'spm_mat_file'),
                                                  ('beta_images', 'beta_images'),
                                                  ('residual_image', 'residual_
↪image')]),
                   ])

```

Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipyre.pipeline.engine.Pipeline.Run` function needs to be called.

```
if __name__ == '__main__':  
    l1pipeline.run('MultiProc')  
    l2pipeline.run('MultiProc')
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipyype source distribution under the `examples` directory.

fmRI: SPM Auditory dataset

Introduction

The `fmri_spm_auditory.py` recreates the classical workflow described in the [SPM8 manual](http://www.fil.ion.ucl.ac.uk/spm/data/auditory/) using auditory dataset that can be downloaded from <http://www.fil.ion.ucl.ac.uk/spm/data/auditory/>:

```
python fmri_spm_auditory.py
```

Import necessary modules from `nipype`.

```
from builtins import range

import nipype.interfaces.io as nio           # Data i/o
import nipype.interfaces.spm as spm         # spm
import nipype.interfaces.fsl as fsl         # fsl
import nipype.interfaces.matlab as mlab     # how to run matlab
import nipype.interfaces.fsl as fsl         # fsl
import nipype.interfaces.utility as util    # utility
import nipype.pipeline.engine as pe        # pypeline engine
import nipype.algorithms.modelgen as model  # model specification
import os                                   # system functions
```

Preliminaries

```
# Set the way matlab should be called
mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
```

Setting up workflows

In this tutorial we will be setting up a hierarchical workflow for `spm` analysis. This will demonstrate how pre-defined workflows can be setup and shared across users, projects and labs.

Setup preprocessing workflow

This is a generic preprocessing workflow that can be used by different analyses

```
preproc = pe.Workflow(name='preproc')
```

We strongly encourage to use 4D files instead of series of 3D for fMRI analyses for many reasons (cleanness and saving and filesystem inodes are among them). However, the the workflow presented in the SPM8 manual which this tutorial is based on uses 3D files. Therefore we leave converting to 4D as an option. We are using *merge_to_4d* variable, because switching between 3d and 4d requires some additional steps (explained later on). Use `nipyne.interfaces.fsl.Merge` to merge a series of 3D files along the time dimension creating a 4d file.

```
merge_to_4d = True

if merge_to_4d:
    merge = pe.Node(interface=fsl.Merge(), name="merge")
    merge.inputs.dimension = "t"
```

Use `nipyne.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```
realign = pe.Node(interface=spm.Realign(), name="realign")
```

Use `nipyne.interfaces.spm.Coregister` to perform a rigid body registration of the functional data to the structural data.

```
coregister = pe.Node(interface=spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'

segment = pe.Node(interface=spm.Segment(), name="segment")
```

Uncomment the following line for faster execution

```
# segment.inputs.gaussians_per_class = [1, 1, 1, 4]
```

Warp functional and structural data to SPM's T1 template using `nipyne.interfaces.spm.Normalize`. The tutorial data set includes the template image, T1.nii.

```
normalize_func = pe.Node(interface=spm.Normalize(), name="normalize_func")
normalize_func.inputs.jobtype = "write"

normalize_struc = pe.Node(interface=spm.Normalize(), name="normalize_struc")
normalize_struc.inputs.jobtype = "write"
```

Smooth the functional data using `nipyne.interfaces.spm.Smooth`.

```
smooth = pe.Node(interface=spm.Smooth(), name="smooth")
```

write_voxel_sizes is the input of the normalize interface that is recommended to be set to the voxel sizes of the target volume. There is no need to set it manually since we can infer it from data using the following function:

```
def get_vox_dims(volume):
    import nibabel as nb
    from nipyne.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    voxdims = hdr.get_zooms()
    return [float(voxdims[0]), float(voxdims[1]), float(voxdims[2])]
```

Here we are connecting all the nodes together. Notice that we add the merge node only if you choose to use 4D. Also *get_vox_dims* function is passed along the input volume of normalise to set the optimal voxel sizes.

```

if merge_to_4d:
    preproc.connect([(merge, realign, [('merged_file', 'in_files')])])

preproc.connect([(realign, coregister, [('mean_image', 'target')]),
                  (coregister, segment, [('coregistered_source', 'data')]),
                  (segment, normalize_func, [('transformation_mat', 'parameter_file
↪')]),
                  (segment, normalize_struc, [('transformation_mat', 'parameter_
↪file'),
                  ('modulated_input_image', 'apply_to_
↪files'),
                  ('modulated_input_image', get_vox_
↪dims), 'write_voxel_sizes')]),
                  (realign, normalize_func, [('realigned_files', 'apply_to_files'),
                  ('realigned_files', get_vox_dims),
↪'write_voxel_sizes')]),
                  (normalize_func, smooth, [('normalized_files', 'in_files')]),
                  ]))

```

Set up analysis workflow

```
l1analysis = pe.Workflow(name='analysis')
```

Generate SPM-specific design information using `nipyype.interfaces.spm.SpecifyModel`.

```
modelspec = pe.Node(interface=model.SpecifySPMModel(), name="modelspec")
```

Generate a first level SPM.mat file for analysis `nipyype.interfaces.spm.Level1Design`.

```

levelldesign = pe.Node(interface=spm.Level1Design(), name="levelldesign")
levelldesign.inputs.bases = {'hrf': {'derivs': [0, 0]}}

```

Use `nipyype.interfaces.spm.EstimateModel` to determine the parameters of the model.

```

levelleestimate = pe.Node(interface=spm.EstimateModel(), name="levelleestimate")
levelleestimate.inputs.estimate_method = {'Classical': 1}

```

```
threshold = pe.Node(interface=spm.Threshold(), name="threshold")
```

Use `nipyype.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.

```

contrastestimate = pe.Node(interface=spm.EstimateContrast(), name=
↪"contrastestimate")

l1analysis.connect([(modelspec, levelldesign, [('session_info', 'session_info')]),
                  (levelldesign, levelleestimate, [('spm_mat_file', 'spm_mat_file
↪')]),
                  (levelleestimate, contrastestimate, [('spm_mat_file', 'spm_mat_
↪file'),
                  ('beta_images', 'beta_
↪images'),
                  ('residual_image',
↪'residual_image')]),
                  (contrastestimate, threshold, [('spm_mat_file', 'spm_mat_file
↪'),
                  ('spmT_images', 'stat_image
↪')]),
                  ]))

```

Preproc + Analysis pipeline

```
l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect([(preproc, l1analysis, [('realign.realignment_parameters',
                                             'modelspec.realignment_parameters'
↪')])])])
```

Plugging in *functional_runs* is a bit more complicated, because model spec expects a list of *runs*. Every run can be a 4D file or a list of 3D files. Therefore for 3D analysis we need a list of lists and to make one we need a helper function.

```
if merge_to_4d:
    l1pipeline.connect([(preproc, l1analysis, [('smooth.smoothed_files',
                                                'modelspec.functional_runs')])])
else:
    def makelist(item):
        return [item]
    l1pipeline.connect([(preproc, l1analysis, [(['smooth.smoothed_files',
↪makelist),
                                                'modelspec.functional_runs')])])])
```

Data specific components

In this tutorial there is only one subject *M00223*.

Below we set some variables to inform the *datasource* about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the *datasource* node in the pipeline.

```
# Specify the location of the data downloaded from http://www.fil.ion.ucl.ac.uk/
↪spm/data/auditory/
data_dir = os.path.abspath('spm_auditory_data')
# Specify the subject directories
subject_list = ['M00223']
# Map field names to individual subject runs.
info = dict(func=[('f', 'subject_id', 'f', 'subject_id', list(range(16, 100)))],
            struct=[('s', 'subject_id', 's', 'subject_id', 2)])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
↪name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The *datasource* attribute *iterables* tells the pipeline engine that it should repeat the analysis on each of the items in the *subject_list*. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in *subject_list*.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a *nipyne.interfaces.io.DataGrabber* object and fill in the information from above about the layout of our data. The *nipyne.pipeline.NodeWrapper* module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                                outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.base_directory = data_dir
```



```
datasource.inputs.template = '%s%s/%s%s_%03d.img'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Experimental paradigm specific components

Here we create a structure that provides information about the experimental paradigm. This is used by the `nipyype.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix.

```
from nipyype.interfaces.base import Bunch
subjectinfo = [Bunch(conditions=['Task'],
                      onsets=[list(range(6, 84, 12))],
                      durations=[[6]])]
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name, Stat, [list of condition names], [weights on those conditions]]. The condition names must match the *names* listed in the *subjectinfo* function described above.

```
cont1 = ('active > rest', 'T', ['Task'], [1])
contrasts = [cont1]

# set up node specific inputs
modelspecref = llpipeline.inputs.analysis.modelspec
modelspecref.input_units = 'scans'
modelspecref.output_units = 'scans'
modelspecref.time_repetition = 7
modelspecref.high_pass_filter_cutoff = 120

l1designref = llpipeline.inputs.analysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition

llpipeline.inputs.preproc.smooth.fwhm = [6, 6, 6]
llpipeline.inputs.analysis.modelspec.subject_info = subjectinfo
llpipeline.inputs.analysis.contrastestimate.contrasts = contrasts
llpipeline.inputs.analysis.threshold.contrast_index = 1
```

Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the `nipyype.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use *workdir* as the disk location to use when running the processes and keeping their outputs. The *use_parameterized_dirs* tells the engine to create sub-directories under *workdir* corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The `nipyype.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

```
level1 = pe.Workflow(name="level1")
level1.base_dir = os.path.abspath('spm_auditory_tutorial/workingdir')

level1.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
               (datasource, llpipeline, [('struct', 'preproc.coregister.source
→'])])
```

```
    ])
if merge_to_4d:
    levell1.connect([(datasource, llpipeline, [('func', 'preproc.merge.in_files
↪')])])
else:
    levell1.connect([(datasource, llpipeline, [('func', 'preproc.realign.in_files
↪')])])
```

Setup storage results

Use `nipyre.interfaces.io.DataSink` to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them.

The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form `directory_name[@subdir]` where parts between `[]` are optional. For example `'realign.@mean'` below creates a directory called `realign` in `'lloutput/subject_id/'` and stores the mean image output from the `Realign` process in the `realign` directory. If the `@` is left out, then a sub-directory with the name `'mean'` would be created and the mean image would be copied to that directory.

```
datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('spm_auditory_tutorial/lloutput')

def getstripdir(subject_id):
    import os
    return os.path.join(os.path.abspath('spm_auditory_tutorial/workingdir'), '_
↪subject_id_%s' % subject_id)

# store relevant outputs from various stages of the 1st level analysis
levell1.connect([(infosource, datasink, [('subject_id', 'container'),
↪('subject_id', getstripdir), 'strip_dir
↪')]),
                (llpipeline, datasink, [('analysis.contrastestimate.con_images',
↪'contrasts.@con'),
↪('analysis.contrastestimate.spmT_images',
↪'contrasts.@T')]),
                ])
```

Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipyre.pipeline.engine.Pipeline.Run` function needs to be called.

```
if __name__ == '__main__':
    levell1.run()
    levell1.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipyre source distribution under the `examples` directory.

fMRI: DARTEL, SPM

The `fmri_spm_dartel.py` integrates several interfaces to perform a first and second level analysis on a two-subject data set. The tutorial can be found in the examples folder. Run the tutorial from inside the nipy tutorial directory:

```
python fmri_spm_dartel.py
```

Import necessary modules from nipy.

```
from __future__ import print_function
from builtins import str
from builtins import range

import nipy.interfaces.io as nio          # Data i/o
import nipy.interfaces.spm as spm        # spm
import nipy.workflows.fmri.spm as spm_wf # spm
import nipy.interfaces.fsl as fsl        # fsl
from nipy.interfaces import utility as niu # Utilities
import nipy.pipeline.engine as pe        # pipeline engine
import nipy.algorithms.rapidart as ra    # artifact detection
import nipy.algorithms.modelgen as model # model specification
import os                                # system functions
```

Preliminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does not handle compressed NIFTI.

```
# Tell fsl to generate all output in uncompressed nifti format
fsl.FSLCommand.set_default_output_type('NIFTI')

# Set the way matlab should be called
# mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
# mlab.MatlabCommand.set_default_paths('/software/spm8')
```

Setting up workflows

In this tutorial we will be setting up a hierarchical workflow for spm analysis. This will demonstrate how pre-defined workflows can be setup and shared across users, projects and labs.

Setup preprocessing workflow

This is a generic preprocessing workflow that can be used by different analyses

```
preproc = pe.Workflow(name='preproc')
```

Use `nipyype.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```
realign = pe.Node(spm.Realign(), name="realign")
realign.inputs.register_to_mean = True
```

Use `nipyype.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```
art = pe.Node(ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```

Skull strip structural images using `nipyype.interfaces.fsl.BET`.

```
skullstrip = pe.Node(fsl.BET(), name="skullstrip")
skullstrip.inputs.mask = True
```

Use `nipyype.interfaces.spm.Coregister` to perform a rigid body registration of the functional data to the structural data.

```
coregister = pe.Node(spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
```

Normalize and smooth functional data using DARTEL template

```
normalize_and_smooth_func = pe.Node(spm.DARTELNorm2MNI(modulate=True), name=
    ↪ 'normalize_and_smooth_func')
fwhmlist = [4]
normalize_and_smooth_func.iterables = ('fwhm', fwhmlist)
```

Normalize structural data using DARTEL template

```
normalize_struct = pe.Node(spm.DARTELNorm2MNI(modulate=True), name='normalize_
    ↪ struct')
normalize_struct.inputs.fwhm = 2

preproc.connect([(realign, coregister, [('mean_image', 'source'),
    ↪ ('realigned_files', 'apply_to_files')]),
    (coregister, normalize_and_smooth_func, [('coregistered_files',
    ↪ 'apply_to_files')]),
    (normalize_struct, skullstrip, [('normalized_files', 'in_file
    ↪ ')]),
    (realign, art, [('realignment_parameters', 'realignment_
    ↪ parameters')]),
    (normalize_and_smooth_func, art, [('normalized_files',
    ↪ 'realigned_files')]),
```

```
(skullstrip, art, [('mask_file', 'mask_file')]),
])
```

Set up analysis workflow

```
l1analysis = pe.Workflow(name='analysis')
```

Generate SPM-specific design information using `nipy.interfaces.spm.SpecifyModel`.

```
modelspec = pe.Node(spm.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = True
```

Generate a first level SPM.mat file for analysis `nipy.interfaces.spm.Level1Design`.

```
level1design = pe.Node(spm.Level1Design(), name="level1design")
level1design.inputs.bases = {'hrf': {'derivs': [0, 0]}}
```

Use `nipy.interfaces.spm.EstimateModel` to determine the parameters of the model.

```
level1estimate = pe.Node(spm.EstimateModel(), name="level1estimate")
level1estimate.inputs.estimate_method = {'Classical': 1}
```

Use `nipy.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.

```
contrastestimate = pe.Node(spm.EstimateContrast(), name="contrastestimate")
```

Use `:class: nipy.interfaces.utility.Select` to select each contrast for reporting.

```
selectcontrast = pe.Node(niu.Select(), name="selectcontrast")
```

Use `nipy.interfaces.fsl.Overlay` to combine the statistical output of the contrast estimate and a background image into one volume.

```
overlaystats = pe.Node(fsl.Overlay(), name="overlaystats")
overlaystats.inputs.stat_thresh = (3, 10)
overlaystats.inputs.show_negative_stats = True
overlaystats.inputs.auto_thresh_bg = True
```

Use `nipy.interfaces.fsl.Slicer` to create images of the overlaid statistical volumes for a report of the first-level results.

```
slicestats = pe.Node(fsl.Slicer(), name="slicestats")
slicestats.inputs.all_axial = True
slicestats.inputs.image_width = 750

l1analysis.connect([(modelspec, level1design, [('session_info', 'session_info')]),
                    (level1design, level1estimate, [('spm_mat_file', 'spm_mat_file'
→])),
                    (level1estimate, contrastestimate, [('spm_mat_file', 'spm_mat_
→file'),
                                                         ('beta_images', 'beta_
→images'),
                                                         ('residual_image',
→'residual_image')]),
                    (contrastestimate, selectcontrast, [('spmT_images', 'inlist
→')]),
                    (selectcontrast, overlaystats, [('out', 'stat_image')]),
```

```
(overlaystats, slicestats, [('out_file', 'in_file')])
])
```

Preproc + Analysis pipeline

```
l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect([(preproc, l1analysis, [('realigned.realignment_parameters',
                                             'modelspec.realignment_parameters'),
                                             ('normalize_and_smooth_func.normalized_
↪files',
                                             'modelspec.functional_runs'),
                                             ('art.outlier_files',
                                             'modelspec.outlier_files'),
                                             ('skullstrip.mask_file',
                                             'level1design.mask_image'),
                                             ('normalize_struct.normalized_files',
                                             'overlaystats.background_image')])],
                    ])
```

Data specific components

The nipy tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume named struct.nii.

Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline.

In the example below, run 'f3' is of type 'func' and gets mapped to a nifti filename through a template '%s.nii'. So 'f3' would become 'f3.nii'.

```
# Specify the location of the data.
# data_dir = os.path.abspath('data')
# Specify the subject directories
subject_list = ['s1', 's3']
# Map field names to individual subject runs.
info = dict(func=[('subject_id', ['f3', 'f5', 'f7', 'f10'])],
            struct=[('subject_id', 'struct')])

infosource = pe.Node(niu.IdentityInterface(fields=['subject_id']), name=
↪"infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a nipy.interfaces.io.DataGrabber object and fill in the information from above about the layout of our data. The nipy.pipeline.NodeWrapper module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
inputnode = pe.Node(niu.IdentityInterface(fields=['in_data']), name='inputnode')
datasource = pe.Node(nio.DataGrabber(infields=['subject_id'],
```

```

                                outfield=[ 'func', 'struct']),
    name='datasource')
datasource.inputs.template = 'nipype-tutorial/data/%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

```

We need to create a separate workflow to make the DARTEL template

```

datasource_dartel = pe.MapNode(nio.DataGrabber(infields=[ 'subject_id'],
                                                outfield=[ 'struct']),
                                name='datasource_dartel',
                                iterfield=[ 'subject_id'])
datasource_dartel.inputs.template = 'nipype-tutorial/data/%s/%s.nii'
datasource_dartel.inputs.template_args = dict(struct=[ 'subject_id', 'struct'])
datasource_dartel.inputs.sort_filelist = True
datasource_dartel.inputs.subject_id = subject_list

```

Here we make sure that struct files have names corresponding to the subject ids. This way we will be able to pick the right field flows later.

```

rename_dartel = pe.MapNode(niu.Rename(format_string="subject_id_%(subject_id)s_
→struct"),
                            iterfield=[ 'in_file', 'subject_id'],
                            name='rename_dartel')
rename_dartel.inputs.subject_id = subject_list
rename_dartel.inputs.keep_ext = True

dartel_workflow = spm_wf.create_DARTEL_template(name='dartel_workflow')
dartel_workflow.inputs.inputs.spec.template_prefix = "template"

```

This function will allow to pick the right field flow for each subject

```

def pickFieldFlow(dartel_flow_fields, subject_id):
    from nipype.utils.filemanip import split_filename
    for f in dartel_flow_fields:
        _, name, _ = split_filename(f)
        if name.find("subject_id_%s" % subject_id):
            return f

    raise Exception

pick_flow = pe.Node(niu.Function(input_names=[ 'dartel_flow_fields',
                                                'subject_id'],
                                output_names=[ 'dartel_flow_field'],
                                function=pickFieldFlow),
                    name="pick_flow")

```

Experimental paradigm specific components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the `nipype.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant.

```

def subjectinfo(subject_id):
    from nipype.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: %s\n" % str(subject_id))
    output = []

```

```
names = ['Task-Odd', 'Task-Even']
for r in range(4):
    onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
    output.insert(r,
                  Bunch(conditions=names,
                        onsets=deepcopy(onsets),
                        durations=[[15] for s in names],
                        amplitudes=None,
                        tmod=None,
                        pmod=None,
                        regressor_names=None,
                        regressors=None))

return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]]. The condition names must match the *names* listed in the *subjectinfo* function described above.

```
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]

# set up node specific inputs
modelspecref = llpipeline.inputs.analysis.modelspec
modelspecref.input_units = 'secs'
modelspecref.output_units = 'secs'
modelspecref.time_repetition = 3.
modelspecref.high_pass_filter_cutoff = 120

l1designref = llpipeline.inputs.analysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition

llpipeline.inputs.analysis.contrastestimate.contrasts = contrasts

# Iterate over each contrast and create report images.
selectcontrast.iterables = ('index', [[i] for i in range(len(contrasts))])
```

Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the `nipyne.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use *workdir* as the disk location to use when running the processes and keeping their outputs. The *use_parameterized_dirs* tells the engine to create sub-directories under *workdir* corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The `nipyne.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

```
level1 = pe.Workflow(name="level1")
level1.base_dir = os.path.abspath('spm_dartel_tutorial/workingdir')
```



```

level1.connect([(inputnode, datasource, [('in_data', 'base_directory')]),
                (inputnode, datasource_dartel, [('in_data', 'base_directory')]),
                (datasource_dartel, rename_dartel, [('struct', 'in_file')]),
                (rename_dartel, dartel_workflow, [('out_file', 'inputspec.
↳structural_files')]),

                (infosource, datasource, [('subject_id', 'subject_id')]),
                (datasource, llpipeline, [('func', 'preproc.realign.in_files',
                ('struct', 'preproc.coregister.target',
                ('struct', 'preproc.normalize_struct.
↳apply_to_files')]),
                (dartel_workflow, llpipeline, [('outputspec.template_file',
↳'preproc.normalize_struct.template_file',
                ('outputspec.template_file',
↳'preproc.normalize_and_smooth_func.template_file')]),
                (infosource, pick_flow, [('subject_id', 'subject_id')]),
                (dartel_workflow, pick_flow, [('outputspec.flow_fields', 'dartel_
↳flow_fields')]),
                (pick_flow, llpipeline, [('dartel_flow_field', 'preproc.normalize_
↳struct.flowfield_files',
                ('dartel_flow_field', 'preproc.normalize_
↳and_smooth_func.flowfield_files')]),
                (infosource, llpipeline, [([('subject_id', subjectinfo),
                ('analysis.modelspec.subject_info')]),
                ])

```

Setup storage results

Use `nipyre.interfaces.io.DataSink` to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them.

The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form `directory_name[.@]subdir` where parts between `[]` are optional. For example `'realign.@mean'` below creates a directory called `realign` in `'lloutput/subject_id/'` and stores the mean image output from the Realign process in the `realign` directory. If the `@` is left out, then a sub-directory with the name `'mean'` would be created and the mean image would be copied to that directory.

```

datasink = pe.Node(nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('spm_dartel_tutorial/lloutput')
report = pe.Node(nio.DataSink(), name='report')
report.inputs.base_directory = os.path.abspath('spm_dartel_tutorial/report')
report.inputs.parameterization = False

def getstripdir(subject_id):
    import os
    return os.path.join(os.path.abspath('spm_dartel_tutorial/workingdir'), '_
↳subject_id_%s' % subject_id)

# store relevant outputs from various stages of the 1st level analysis
level1.connect([(infosource, datasink, [('subject_id', 'container'),
                ('subject_id', getstripdir), 'strip_dir
↳')]),
                (llpipeline, datasink, [('analysis.contrastestimate.con_images',
↳'contrasts.@con',
                ('analysis.contrastestimate.spmT_images',
↳'contrasts.@T')]),

```

```

        (infosource, report, [('subject_id', 'container'),
                               (('subject_id', getstripdir), 'strip_dir
→')])),
        (l1pipeline, report, [('analysis.slicestats.out_file', '@report
→')])),
    ])

```

Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipyype.pipeline.engine.Pipeline.Run` function needs to be called.

```

if __name__ == '__main__':
    level1.run(plugin_args={'n_procs': 4})
    level1.write_graph()

```

Setup level 2 pipeline

Use `nipyype.interfaces.io.DataGrabber` to extract the contrast images across a group of first level subjects. Unlike the previous pipeline that iterated over subjects, this pipeline will iterate over contrasts.

```

# collect all the con images for each contrast.
contrast_ids = list(range(1, len(contrasts) + 1))
l2source = pe.Node(nio.DataGrabber(infields=['fwhm', 'con']), name="l2source")
# we use .*i* to capture both .img (SPM8) and .nii (SPM12)
l2source.inputs.template = os.path.abspath('spm_dartel_tutorial/l1output/*/con*/*/
→_fwhm_%d/con_%04d.*i*')
# iterate over all contrast images
l2source.iterables = [('fwhm', fwhmlist),
                     ('con', contrast_ids)]
l2source.inputs.sort_filelist = True

```

Use `nipyype.interfaces.spm.OneSampleTTestDesign` to perform a simple statistical analysis of the contrasts from the group of subjects (n=2 in this example).

```

# setup a 1-sample t-test node
onesampttestdes = pe.Node(spm.OneSampleTTestDesign(), name="onesampttestdes")
l2estimate = pe.Node(spm.EstimateModel(), name="level2estimate")
l2estimate.inputs.estimate_method = {'Classical': 1}
l2conestimate = pe.Node(spm.EstimateContrast(), name="level2conestimate")
cont1 = ('Group', 'T', ['mean'], [1])
l2conestimate.inputs.contrasts = [cont1]
l2conestimate.inputs.group_contrast = True

```

As before, we setup a pipeline to connect these two nodes (l2source -> onesampttest).

```

l2pipeline = pe.Workflow(name="level2")
l2pipeline.base_dir = os.path.abspath('spm_dartel_tutorial/l2output')
l2pipeline.connect([(l2source, onesampttestdes, [('outfiles', 'in_files')]),
                    (onesampttestdes, l2estimate, [('spm_mat_file', 'spm_mat_
→file')]),
                    (l2estimate, l2conestimate, [('spm_mat_file', 'spm_mat_file',
                    ('beta_images', 'beta_images'),
                    ('residual_image', 'residual_
→image')])]),

```

```
l)
```

Execute the second level pipeline

```
if __name__ == '__main__':  
    l2pipeline.run()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

fMRI: Famous vs non-famous faces, SPM

Introduction

The `fmri_spm_face.py` recreates the classical workflow described in the [SPM8 manual](http://www.fil.ion.ucl.ac.uk/spm/data/face_rep/) using face dataset that can be downloaded from http://www.fil.ion.ucl.ac.uk/spm/data/face_rep/:

```
python fmri_spm.py
```

Import necessary modules from `nipype`.

```
from __future__ import division
from builtins import range

import os                                # system functions
import nipype.interfaces.io as nio       # Data i/o
import nipype.interfaces.spm as spm      # spm
import nipype.interfaces.matlab as mlab  # how to run matlab
import nipype.interfaces.utility as util # utility
import nipype.pipeline.engine as pe      # pypeline engine
import nipype.algorithms.modelgen as model # model specification
```

Preliminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does not handle compressed NIFTI.

```
# Set the way matlab should be called
mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
# If SPM is not in your MATLAB path you should add it here
# mlab.MatlabCommand.set_default_paths('/path/to/your/spm8')
```

Setting up workflows

In this tutorial we will be setting up a hierarchical workflow for `spm` analysis. It one is slightly different then the one used in `spm_tutorial2`.

Setup preprocessing workflow

This is a generic preprocessing workflow that can be used by different analyses

```
preproc = pe.Workflow(name='preproc')
```

Use `nipyype.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```
realign = pe.Node(interface=spm.Realign(), name="realign")

slice_timing = pe.Node(interface=spm.SliceTiming(), name="slice_timing")
```

Use `nipyype.interfaces.spm.Coregister` to perform a rigid body registration of the functional data to the structural data.

```
coregister = pe.Node(interface=spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'

segment = pe.Node(interface=spm.Segment(), name="segment")
segment.inputs.save_bias_corrected = True
```

Uncomment the following line for faster execution

```
# segment.inputs.gaussians_per_class = [1, 1, 1, 4]
```

Warp functional and structural data to SPM's T1 template using `nipyype.interfaces.spm.Normalize`. The tutorial data set includes the template image, T1.nii.

```
normalize_func = pe.Node(interface=spm.Normalize(), name="normalize_func")
normalize_func.inputs.jobtype = "write"

normalize_struc = pe.Node(interface=spm.Normalize(), name="normalize_struc")
normalize_struc.inputs.jobtype = "write"
```

Smooth the functional data using `nipyype.interfaces.spm.Smooth`.

```
smooth = pe.Node(interface=spm.Smooth(), name="smooth")
```

`write_voxel_sizes` is the input of the normalize interface that is recommended to be set to the voxel sizes of the target volume. There is no need to set it manually since we can infer it from data using the following function:

```
def get_vox_dims(volume):
    import nibabel as nb
    from nipyype.utils import NUMPY_MMAP
    if isinstance(volume, list):
        volume = volume[0]
    nii = nb.load(volume, mmap=NUMPY_MMAP)
    hdr = nii.header
    voxdims = hdr.get_zooms()
    return [float(voxdims[0]), float(voxdims[1]), float(voxdims[2])]
```

Here we are connecting all the nodes together. Notice that we add the merge node only if you choose to use 4D. Also `get_vox_dims` function is passed along the input volume of `normalize` to set the optimal voxel sizes.

```
preproc.connect([(realign, coregister, [('mean_image', 'target')]),
                 (coregister, segment, [('coregistered_source', 'data')]),
                 (segment, normalize_func, [('transformation_mat', 'parameter_file',
→])),
                 (segment, normalize_struc, [('transformation_mat', 'parameter_
→file'),
                                             ('bias_corrected_image', 'apply_to_
→files')],
```

```

                                                    (('bias_corrected_image', get_vox_
↪dims), 'write_voxel_sizes'])),
                (realign, slice_timing, [('realigned_files', 'in_files')]),
                (slice_timing, normalize_func, [('timecorrected_files', 'apply_
↪to_files'),
                                                    (('timecorrected_files', get_vox_
↪dims), 'write_voxel_sizes'])),
                (normalize_func, smooth, [('normalized_files', 'in_files')]),
                ])

```

Set up analysis workflow

```
l1analysis = pe.Workflow(name='analysis')
```

Generate SPM-specific design information using `nipy.interfaces.spm.SpecifyModel`.

```
modelspec = pe.Node(interface=model.SpecifySPMModel(), name="modelspec")
```

Generate a first level SPM.mat file for analysis `nipy.interfaces.spm.Level1Design`.

```
levelldesign = pe.Node(interface=spm.Level1Design(), name="levelldesign")
```

Use `nipy.interfaces.spm.EstimateModel` to determine the parameters of the model.

```
levellestimate = pe.Node(interface=spm.EstimateModel(), name="levellestimate")
levellestimate.inputs.estimate_method = {'Classical': 1}
```

```
threshold = pe.Node(interface=spm.Threshold(), name="threshold")
```

Use `nipy.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.

```

contrastestimate = pe.Node(interface=spm.EstimateContrast(), name=
↪"contrastestimate")

def pickfirst(l):
    return l[0]

l1analysis.connect([(modelspec, levelldesign, [('session_info', 'session_info')]),
                    (levelldesign, levellestimate, [('spm_mat_file', 'spm_mat_file
↪')]),
                    (levellestimate, contrastestimate, [('spm_mat_file', 'spm_mat_
↪file'),
                                                            ('beta_images', 'beta_
↪images'),
                                                            ('residual_image',
↪'residual_image')]),
                    (contrastestimate, threshold, [('spm_mat_file', 'spm_mat_file
↪'),
                                                    (('spmT_images', pickfirst),
↪'stat_image')]),
                    ])

```

Preproc + Analysis pipeline

```
l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect([(preproc, l1analysis, [('realign.realignment_parameters',
↪ 'modelspec.realignment_parameters
↪ ')])])])
```

Plugging in *functional_runs* is a bit more complicated, because model spec expects a list of *runs*. Every run can be a 4D file or a list of 3D files. Therefore for 3D analysis we need a list of lists and to make one we need a helper function.

```
def makelist(item):
    return [item]
l1pipeline.connect([(preproc, l1analysis, [(['smooth.smoothed_files', makelist),
↪ 'modelspec.functional_runs'])])])
```

Data specific components

In this tutorial there is only one subject *M03953*.

Below we set some variables to inform the *datasource* about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the *datasource* node in the pipeline.

```
# Specify the location of the data downloaded from http://www.fil.ion.ucl.ac.uk/
↪spm/data/face_rep/face_rep_SPM5.html
data_dir = os.path.abspath('spm_face_data')
# Specify the subject directories
subject_list = ['M03953']
# Map field names to individual subject runs.
info = dict(func=[('RawEPI', 'subject_id', 5, ["_%04d" % i for i in range(6,
↪357)])]),
            struct=[('Structural', 'subject_id', 7, '')])

infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                    name="infosource")
```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The *datasource* attribute *iterables* tells the pipeline engine that it should repeat the analysis on each of the items in the *subject_list*. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in *subject_list*.

```
infosource.iterables = ('subject_id', subject_list)
```

Now we create a *nipy.interfaces.io.DataGrabber* object and fill in the information from above about the layout of our data. The *nipy.pipeline.NodeWrapper* module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                              outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/s%s_%04d%s.img'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```


Experimental paradigm specific components

Here we create a structure that provides information about the experimental paradigm. This is used by the `nipyne.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix.

```
from nipyne.interfaces.base import Bunch
```

We're importing the onset times from a mat file (found on http://www.fil.ion.ucl.ac.uk/spm/data/face_rep/)

```
from scipy.io.matlab import loadmat
mat = loadmat(os.path.join(data_dir, "sots.mat"), struct_as_record=False)
sot = mat['sot'][0]
itemlag = mat['itemlag'][0]

subjectinfo = [Bunch(conditions=['N1', 'N2', 'F1', 'F2'],
                        onsets=[sot[0], sot[1], sot[2], sot[3]],
                        durations=[[0], [0], [0], [0]],
                        amplitudes=None,
                        tmod=None,
                        pmod=None,
                        regressor_names=None,
                        regressors=None)]
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name, Stat, [list of condition names], [weights on those conditions]]. The condition names must match the *names* listed in the *subjectinfo* function described above.

```
cond1 = ('positive effect of condition', 'T', ['N1*bf(1)', 'N2*bf(1)', 'F1*bf(1)',
↪ 'F2*bf(1)'], [1, 1, 1, 1])
cond2 = ('positive effect of condition_dtemp', 'T', ['N1*bf(2)', 'N2*bf(2)',
↪ 'F1*bf(2)', 'F2*bf(2)'], [1, 1, 1, 1])
cond3 = ('positive effect of condition_ddisp', 'T', ['N1*bf(3)', 'N2*bf(3)',
↪ 'F1*bf(3)', 'F2*bf(3)'], [1, 1, 1, 1])
# non-famous > famous
fam1 = ('positive effect of Fame', 'T', ['N1*bf(1)', 'N2*bf(1)', 'F1*bf(1)',
↪ 'F2*bf(1)'], [1, 1, -1, -1])
fam2 = ('positive effect of Fame_dtemp', 'T', ['N1*bf(2)', 'N2*bf(2)', 'F1*bf(2)',
↪ 'F2*bf(2)'], [1, 1, -1, -1])
fam3 = ('positive effect of Fame_ddisp', 'T', ['N1*bf(3)', 'N2*bf(3)', 'F1*bf(3)',
↪ 'F2*bf(3)'], [1, 1, -1, -1])
# rep1 > rep2
rep1 = ('positive effect of Rep', 'T', ['N1*bf(1)', 'N2*bf(1)', 'F1*bf(1)',
↪ 'F2*bf(1)'], [1, -1, 1, -1])
rep2 = ('positive effect of Rep_dtemp', 'T', ['N1*bf(2)', 'N2*bf(2)', 'F1*bf(2)',
↪ 'F2*bf(2)'], [1, -1, 1, -1])
rep3 = ('positive effect of Rep_ddisp', 'T', ['N1*bf(3)', 'N2*bf(3)', 'F1*bf(3)',
↪ 'F2*bf(3)'], [1, -1, 1, -1])
int1 = ('positive interaction of Fame x Rep', 'T', ['N1*bf(1)', 'N2*bf(1)',
↪ 'F1*bf(1)', 'F2*bf(1)'], [-1, -1, -1, 1])
int2 = ('positive interaction of Fame x Rep_dtemp', 'T', ['N1*bf(2)', 'N2*bf(2)',
↪ 'F1*bf(2)', 'F2*bf(2)'], [1, -1, -1, 1])
int3 = ('positive interaction of Fame x Rep_ddisp', 'T', ['N1*bf(3)', 'N2*bf(3)',
↪ 'F1*bf(3)', 'F2*bf(3)'], [1, -1, -1, 1])

contf1 = ['average effect condition', 'F', [cond1, cond2, cond3]]
contf2 = ['main effect Fam', 'F', [fam1, fam2, fam3]]
contf3 = ['main effect Rep', 'F', [rep1, rep2, rep3]]
contf4 = ['interaction: Fam x Rep', 'F', [int1, int2, int3]]
contrasts = [cond1, cond2, cond3, fam1, fam2, fam3, rep1, rep2, rep3, int1, int2,
↪ int3, contf1, contf2, contf3, contf4]
```

Setting up nodes inputs

```

num_slices = 24
TR = 2.

slice_timingref = l1pipeline.inputs.preproc.slice_timing
slice_timingref.num_slices = num_slices
slice_timingref.time_repetition = TR
slice_timingref.time_acquisition = TR - TR / float(num_slices)
slice_timingref.slice_order = list(range(num_slices, 0, -1))
slice_timingref.ref_slice = int(num_slices / 2)

l1pipeline.inputs.preproc.smooth.fwhm = [8, 8, 8]

# set up node specific inputs
modelspecref = l1pipeline.inputs.analysis.modelspec
modelspecref.input_units = 'scans'
modelspecref.output_units = 'scans'
modelspecref.time_repetition = TR
modelspecref.high_pass_filter_cutoff = 120

l1designref = l1pipeline.inputs.analysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition
l1designref.microtime_resolution = slice_timingref.num_slices
l1designref.microtime_onset = slice_timingref.ref_slice
l1designref.bases = {'hrf': {'derivs': [1, 1]}}

```

The following lines automatically inform SPM to create a default set of contrasts for a factorial design.

```

# l1designref.factor_info = [dict(name = 'Fame', levels = 2),
#                           dict(name = 'Rep', levels = 2)]

l1pipeline.inputs.analysis.modelspec.subject_info = subjectinfo
l1pipeline.inputs.analysis.contrastestimate.contrasts = contrasts
l1pipeline.inputs.analysis.threshold.contrast_index = 1

```

Use derivative estimates in the non-parametric model

```

l1pipeline.inputs.analysis.contrastestimate.use_derivs = True

```

Setting up parametricvariation of the model

```

subjectinfo_param = [Bunch(conditions=['N1', 'N2', 'F1', 'F2'],
                             onsets=[sot[0], sot[1], sot[2], sot[3]],
                             durations=[[0], [0], [0], [0]],
                             amplitudes=None,
                             tmod=None,
                             pmod=[None,
                                     Bunch(name=['Lag'],
                                             param=itemlag[1].tolist(),
                                             poly=[2]),
                                     None,
                                     Bunch(name=['Lag'],
                                             param=itemlag[3].tolist(),
                                             poly=[2])),
                             regressor_names=None,
                             regressors=None)]

```

```

cont1 = ('Famous_lag1', 'T', ['F2xLag^1'], [1])
cont2 = ('Famous_lag2', 'T', ['F2xLag^2'], [1])
fcont1 = ('Famous Lag', 'F', [cont1, cont2])
paramcontrasts = [cont1, cont2, fcont1]

paramanalysis = llanalysis.clone(name='paramanalysis')

paramanalysis.inputs.level1design.bases = {'hrf': {'derivs': [0, 0]}}
paramanalysis.inputs.modelspec.subject_info = subjectinfo_param
paramanalysis.inputs.contrastestimate.contrasts = paramcontrasts
paramanalysis.inputs.contrastestimate.use_derivs = False

llpipeline.connect([(preproc, paramanalysis, [('realign.realignment_parameters',
↪                                     'modelspec.realignment_parameters
↪                                     'smooth.smoothed_files',
↪                                     'modelspec.functional_runs')]))])

```

Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the `nipyype.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use *workdir* as the disk location to use when running the processes and keeping their outputs. The *use_parameterized_dirs* tells the engine to create sub-directories under *workdir* corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The `nipyype.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

```

level1 = pe.Workflow(name="level1")
level1.base_dir = os.path.abspath('spm_face_tutorial/workingdir')

level1.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                (datasource, llpipeline, [('struct', 'preproc.coregister.source'),
                ('func', 'preproc.realign.in_files')])
                ])

```

Setup storage results

Use `nipyype.interfaces.io.DataSink` to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them.

The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form `directory_name[.@subdir]` where parts between `[]` are optional. For example `'realign.@mean'` below creates a directory called `realign` in `'lloutput/subject_id/'` and stores the mean image output from the `Realign` process in the `realign` directory. If the `@` is left out, then a sub-directory with the name `'mean'` would be created and the mean image would be copied to that directory.

```

datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('spm_auditory_tutorial/lloutput')

def getstripdir(subject_id):

```

```
import os
return os.path.join(os.path.abspath('spm_auditory_tutorial/workingdir'), '_
↳subject_id_%s' % subject_id)

# store relevant outputs from various stages of the 1st level analysis
levell.connect([(infosource, datasink, [('subject_id', 'container'),
↳('subject_id', getstripdir), 'strip_dir
↳')]),
               (l1pipeline, datasink, [('analysis.contrastestimate.con_images',
↳'contrasts.@con'),
               ('analysis.contrastestimate.spmT_images',
↳'contrasts.@T'),
               ('paramanalysis.contrastestimate.con_
↳images', 'paramcontrasts.@con'),
               ('paramanalysis.contrastestimate.spmT_
↳images', 'paramcontrasts.@T')]),
               ])
```

Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipy.pipeline.engine.Pipeline.Run` function needs to be called.

```
if __name__ == '__main__':
    levell.run()
    levell.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipy source distribution under the `examples` directory.

fMRI: SPM nested workflows

The `fmri_spm.py` integrates several interfaces to perform a first and second level analysis on a two-subject data set. The tutorial can be found in the examples folder. Run the tutorial from inside the nipype tutorial directory:

```
python fmri_spm_nested.py
```

Import necessary modules from nipype.

```
from __future__ import print_function
from builtins import str
from builtins import range
import os.path as op                                # system functions

from nipype.interfaces import io as nio             # Data i/o
from nipype.interfaces import spm as spm            # spm
# from nipype.interfaces import matlab as mlab       # how to run matlab
from nipype.interfaces import fsl as fsl            # fsl
from nipype.interfaces import utility as niu        # utility
from nipype.pipeline import engine as pe            # pypeline engine
from nipype.algorithms import rapidart as ra        # artifact detection
from nipype.algorithms import modelgen as model     # model specification
```

Preliminaries

Set any package specific configuration. The output file format for FSL routines is being set to uncompressed NIFTI and a specific version of matlab is being used. The uncompressed format is required because SPM does not handle compressed NIFTI.

```
# Tell fsl to generate all output in uncompressed nifti format
fsl.FSLCommand.set_default_output_type('NIFTI')

# Set the way matlab should be called
# mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
# mlab.MatlabCommand.set_default_paths('/software/spm8')
```

Setting up workflows

In this tutorial we will be setting up a hierarchical workflow for spm analysis. This will demonstrate how pre-defined workflows can be setup and shared across users, projects and labs.

Example of how to inline functions in connect()

```
::
def _template_path(in_data): import os.path as op return op.abspath(op.join(in_data, 'nipyne-
tutorial/data/T1.nii'))
```

Set-up preprocessing workflow

This is a generic preprocessing workflow that can be used by different analyses

```
preproc = pe.Workflow(name='preproc')
```

A node called inputnode is set to designate the path in which input data are located:

```
inputnode = pe.Node(niu.IdentityInterface(fields=['in_data']), name='inputnode')
```

Use `nipyne.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```
realign = pe.Node(spm.Realign(), name="realign")
realign.inputs.register_to_mean = True
```

Use `nipyne.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```
art = pe.Node(ra.ArtifactDetect(), name="art")
art.inputs.use_differences = [True, False]
art.inputs.use_norm = True
art.inputs.norm_threshold = 1
art.inputs.zintensity_threshold = 3
art.inputs.mask_type = 'file'
art.inputs.parameter_source = 'SPM'
```

Skull strip structural images using `nipyne.interfaces.fsl.BET`.

```
skullstrip = pe.Node(fsl.BET(), name="skullstrip")
skullstrip.inputs.mask = True
```

Use `nipyne.interfaces.spm.Coregister` to perform a rigid body registration of the functional data to the structural data.

```
coregister = pe.Node(spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
```

Warp functional and structural data to SPM's T1 template using `nipyne.interfaces.spm.Normalize`. The tutorial data set includes the template image, T1.nii.

```
normalize = pe.Node(spm.Normalize(), name="normalize")
```

Smooth the functional data using `nipyne.interfaces.spm.Smooth`.

```
smooth = pe.Node(spm.Smooth(), name="smooth")
fwhmlist = [4]
smooth.iterables = ('fwhm', fwhmlist)
```

```

preproc.connect([(inputnode, normalize, [(['in_data', '_template_path'), 'template
→']),
                (realign, coregister, [(['mean_image', 'source'),
                                         ('realigned_files', 'apply_to_files')]),
                (coregister, normalize, [(['coregistered_files', 'apply_to_files
→']),
                (normalize, smooth, [(['normalized_files', 'in_files')]),
                (normalize, skullstrip, [(['normalized_source', 'in_file')]),
                (realign, art, [(['realignment_parameters', 'realignment_
→parameters')]),
                (normalize, art, [(['normalized_files', 'realigned_files')]),
                (skullstrip, art, [(['mask_file', 'mask_file')]),
                ]))

```

Set up analysis workflow

```
l1analysis = pe.Workflow(name='analysis')
```

Generate SPM-specific design information using `nipy.interfaces.spm.SpecifyModel`.

```

modelspec = pe.Node(model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = True

```

Generate a first level SPM.mat file for analysis `nipy.interfaces.spm.Level1Design`.

```

levelldesign = pe.Node(spm.Level1Design(), name="levelldesign")
levelldesign.inputs.bases = {'hrf': {'derivs': [0, 0]}}

```

Use `nipy.interfaces.spm.EstimateModel` to determine the parameters of the model.

```

levellestimate = pe.Node(spm.EstimateModel(), name="levellestimate")
levellestimate.inputs.estimate_method = {'Classical': 1}

```

Use `nipy.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.

```
contrastestimate = pe.Node(spm.EstimateContrast(), name="contrastestimate")
```

Use `:class: nipy.interfaces.utility.Select` to select each contrast for reporting.

```
selectcontrast = pe.Node(niu.Select(), name="selectcontrast")
```

Use `nipy.interfaces.fsl.Overlay` to combine the statistical output of the contrast estimate and a background image into one volume.

```

overlaystats = pe.Node(fsl.Overlay(), name="overlaystats")
overlaystats.inputs.stat_thresh = (3, 10)
overlaystats.inputs.show_negative_stats = True
overlaystats.inputs.auto_thresh_bg = True

```

Use `nipy.interfaces.fsl.Slicer` to create images of the overlaid statistical volumes for a report of the first-level results.

```

slicestats = pe.Node(fsl.Slicer(), name="slicestats")
slicestats.inputs.all_axial = True
slicestats.inputs.image_width = 750

l1analysis.connect([(modelspec, levelldesign, [(['session_info', 'session_info')]),
                    (levelldesign, levellestimate, [(['spm_mat_file', 'spm_mat_file
→')]),

```

```

        (levelleestimate, contrastestimate, [('spm_mat_file', 'spm_mat_
↪file'),
        ('beta_images', 'beta_
↪images'),
        ('residual_image',
↪'residual_image'))]),
        (contrastestimate, selectcontrast, [('spmT_images', 'inlist
↪')]),
        (selectcontrast, overlaystats, [('out', 'stat_image')]),
        (overlaystats, slicestats, [('out_file', 'in_file')])
    ])

```

Preproc + Analysis pipeline

```

l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect([(preproc, l1analysis, [('realign.realignment_parameters',
        'modelspec.realignment_parameters'),
        ('smooth.smoothed_files',
        'modelspec.functional_runs'),
        ('art.outlier_files',
        'modelspec.outlier_files'),
        ('skullstrip.mask_file',
        'level1design.mask_image'),
        ('normalize.normalized_source',
        'overlaystats.background_image')]),
        ]))

```

Data specific components

The nipy tutorial contains data for two subjects. Subject data is in two subdirectories, s1 and s2. Each subject directory contains four functional volumes: f3.nii, f5.nii, f7.nii, f10.nii. And one anatomical volume named struct.nii.

Below we set some variables to inform the datasource about the layout of our data. We specify the location of the data, the subject sub-directories and a dictionary that maps each run to a mnemonic (or field) for the run type (struct or func). These fields become the output fields of the datasource node in the pipeline.

In the example below, run 'f3' is of type 'func' and gets mapped to a nifti filename through a template '%s.nii'. So 'f3' would become 'f3.nii'.

```

# Specify the subject directories
subject_list = ['s1', 's3']
# Map field names to individual subject runs.
info = dict(func=[('subject_id', ['f3', 'f5', 'f7', 'f10'])],
            struct=[('subject_id', 'struct')])

infosource = pe.Node(niu.IdentityInterface(fields=['subject_id']), name=
↪"infosource")

```

Here we set up iteration over all the subjects. The following line is a particular example of the flexibility of the system. The datasource attribute iterables tells the pipeline engine that it should repeat the analysis on each of the items in the subject_list. In the current example, the entire first level preprocessing and estimation will be repeated for each subject contained in subject_list.

```

infosource.iterables = ('subject_id', subject_list)

```


Now we create a `nipyne.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipyne.pipeline.NodeWrapper` module wraps the interface object and provides additional housekeeping and pipeline specific functionality.

```
datasource = pe.Node(nio.DataGrabber(infields=['subject_id'],
                                      outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.template = 'nipyne-tutorial/data/%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

Experimental paradigm specific components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the `nipyne.interfaces.spm.SpecifyModel` to create the information necessary to generate an SPM design matrix. In this tutorial, the same paradigm was used for every participant.

```
def subjectinfo(subject_id):
    from nipyne.interfaces.base import Bunch
    from copy import deepcopy
    print("Subject ID: %s\n" % str(subject_id))
    output = []
    names = ['Task-Odd', 'Task-Even']
    for r in range(4):
        onsets = [list(range(15, 240, 60)), list(range(45, 240, 60))]
        output.insert(r,
                      Bunch(conditions=names,
                           onsets=deepcopy(onsets),
                           durations=[[15] for s in names],
                           amplitudes=None,
                           tmod=None,
                           pmod=None,
                           regressor_names=None,
                           regressors=None))

    return output
```

Setup the contrast structure that needs to be evaluated. This is a list of lists. The inner list specifies the contrasts and has the following format - [Name,Stat,[list of condition names],[weights on those conditions]]. The condition names must match the *names* listed in the *subjectinfo* function described above.

```
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrasts = [cont1, cont2]

# set up node specific inputs
modelspecref = llpipeline.inputs.analysis.modelspec
modelspecref.input_units = 'secs'
modelspecref.output_units = 'secs'
modelspecref.time_repetition = 3.
modelspecref.high_pass_filter_cutoff = 120

l1designref = llpipeline.inputs.analysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition

llpipeline.inputs.analysis.contrastestimate.contrasts = contrasts
```

```
# Iterate over each contrast and create report images.
selectcontrast.iterables = ('index', [[i] for i in range(len(contrasts))])
```

Setup the pipeline

The nodes created above do not describe the flow of data. They merely describe the parameters used for each function. In this section we setup the connections between the nodes such that appropriate outputs from nodes are piped into appropriate inputs of other nodes.

Use the `nipy.pipeline.engine.Pipeline` to create a graph-based execution pipeline for first level analysis. The config options tells the pipeline engine to use *workdir* as the disk location to use when running the processes and keeping their outputs. The *use_parameterized_dirs* tells the engine to create sub-directories under *workdir* corresponding to the iterables in the pipeline. Thus for this pipeline there will be subject specific sub-directories.

The `nipy.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of the processing nodes.

```
level1 = pe.Workflow(name="level1")
level1.base_dir = op.abspath('spm_tutorial2/workingdir')

level1.connect([(inputnode, datasource, [('in_data', 'base_directory')]),
                (infosource, datasource, [('subject_id', 'subject_id')]),
                (datasource, llpipeline, [('func', 'preproc.realign.in_files'),
                                          ('struct', 'preproc.coregister.target'),
                                          ('struct', 'preproc.normalize.source
→')]),
                (infosource, llpipeline, (('subject_id', subjectinfo),
                                          'analysis.modelspec.subject_info'))],
                ])

```

Setup storage results

Use `nipy.interfaces.io.DataSink` to store selected outputs from the pipeline in a specific location. This allows the user to selectively choose important output bits from the analysis and keep them.

The first step is to create a datasink node and then to connect outputs from the modules above to storage locations. These take the following form `directory_name[.@subdir]` where parts between `[]` are optional. For example `'realign.@mean'` below creates a directory called `realign` in `'l1output/subject_id/'` and stores the mean image output from the Realign process in the `realign` directory. If the `@` is left out, then a sub-directory with the name `'mean'` would be created and the mean image would be copied to that directory.

```
datasink = pe.Node(nio.DataSink(), name="datasink")
datasink.inputs.base_directory = op.abspath('spm_tutorial2/l1output')
report = pe.Node(nio.DataSink(), name='report')
report.inputs.base_directory = op.abspath('spm_tutorial2/report')
report.inputs.parameterization = False

def getstripdir(subject_id):
    import os.path as op
    return op.join(op.abspath('spm_tutorial2/workingdir'), '_subject_id_%s' %_
→subject_id)

# store relevant outputs from various stages of the 1st level analysis
```

```

level1.connect([(infosource, datasink, [('subject_id', 'container'),
                                         (('subject_id', getstripdir), 'strip_dir
→')])),
                (l1pipeline, datasink, [('analysis.contrastestimate.con_images',
→'contrasts.@con'),
                                         ('analysis.contrastestimate.spmT_images',
→'contrasts.@T')])),
                (infosource, report, [('subject_id', 'container'),
                                       (('subject_id', getstripdir), 'strip_dir
→')])),
                (l1pipeline, report, [('analysis.slicestats.out_file', '@report
→')])),
                ])

```

Execute the pipeline

The code discussed above sets up all the necessary data structures with appropriate parameters and the connectivity between the processes, but does not generate any output. To actually run the analysis on the data the `nipyre.pipeline.engine.Pipeline.Run` function needs to be called.

```

if __name__ == '__main__':
    level1.run('MultiProc')
    level1.write_graph()

```

Setup level 2 pipeline

Use `nipyre.interfaces.io.DataGrabber` to extract the contrast images across a group of first level subjects. Unlike the previous pipeline that iterated over subjects, this pipeline will iterate over contrasts.

```

# collect all the con images for each contrast.
contrast_ids = list(range(1, len(contrasts) + 1))
l2source = pe.Node(nio.DataGrabber(infields=['fwhm', 'con']), name="l2source")
# we use .*i* to capture both .img (SPM8) and .nii (SPM12)
l2source.inputs.template = op.abspath('spm_tutorial2/l1output/*/con*/*_fwhm_%d/
→con_%04d.*i*')
# iterate over all contrast images
l2source.iterables = [('fwhm', fwhmlist),
                     ('con', contrast_ids)]
l2source.inputs.sort_filelist = True

```

Use `nipyre.interfaces.spm.OneSampleTTestDesign` to perform a simple statistical analysis of the contrasts from the group of subjects (n=2 in this example).

```

# setup a 1-sample t-test node
onesamletttestdes = pe.Node(spm.OneSampleTTestDesign(), name="onesampttestdes")
l2estimate = pe.Node(spm.EstimateModel(), name="level2estimate")
l2estimate.inputs.estimate_method = {'Classical': 1}
l2conestimate = pe.Node(spm.EstimateContrast(), name="level2conestimate")
cont1 = ('Group', 'T', ['mean'], [1])
l2conestimate.inputs.contrasts = [cont1]
l2conestimate.inputs.group_contrast = True

```

As before, we setup a pipeline to connect these two nodes (l2source -> onesamletttest).

```

l2pipeline = pe.Workflow(name="level2")
l2pipeline.base_dir = op.abspath('spm_tutorial2/l2output')

```

```
l2pipeline.connect([(l2source, onesamplertestdes, [('outfiles', 'in_files')]),
                    (onesamplertestdes, l2estimate, [('spm_mat_file', 'spm_mat_
↪file')])),
                    (l2estimate, l2conestimate, [('spm_mat_file', 'spm_mat_file'),
                                                  ('beta_images', 'beta_images'),
                                                  ('residual_image', 'residual_
↪image')])),
                    ])
```

Execute the second level pipeline

```
if __name__ == '__main__':
    l2pipeline.run('MultiProc')
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

HOWTO: Using caching without using Workflow

Using nipy in an imperative way: caching without workflow

Note that in the following example, we are calling command-lines with disk I/O that persists across runs, but we never have to worry about the file names or the directories.

The disk location of the persistence is encoded by hashes. To find out where an operation has been persisted, simply look in its output variable:

```
out.runtime.cwd
```

```
from nipy.interfaces import fsl
fsl.FSLCommand.set_default_output_type('NIFTI')

from nipy.caching import Memory

import glob

# First retrieve the list of files that we want to work upon
in_files = glob.glob('data/*/f3.nii')

# Create a memory context
mem = Memory('.')

# Apply an arbitrary (and pointless, here) threshold to the files)
threshold = [mem.cache(fsl.Threshold)(in_file=f, thresh=i)
              for i, f in enumerate(in_files)]

# Merge all these files along the time dimension
out_merge = mem.cache(fsl.Merge)(dimension="t",
                                in_files=[t.outputs.out_file for t in threshold],
                                )

# And finally compute the mean
out_mean = mem.cache(fsl.MeanImage)(in_file=out_merge.outputs.merged_file)

# To avoid having increasing disk size we can keep only what was touched
# in this run
# mem.clear_previous_runs()

# or what wasn't used since the start of 2011
```

```
# mem.clear_runs_since(year=2011)
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipyte source distribution under the `examples` directory.

rsfMRI: ANTS, FS, FSL, SPM, aCompCor

A preprocessing workflow for Siemens resting state data.

This workflow makes use of:

- ANTS
- FreeSurfer
- FSL
- SPM
- CompCor

For example:

```
python rsfmri_preprocessing.py -d /data/12345-34-1.dcm -f /data/Resting.nii
-s subj001 -o output -p PBS --plugin_args "dict(qsub_args='-q many')"
```

or

```
python rsfmri_vol_surface_preprocessing.py -f SUB_1024011/E?/func/rest.nii
-t OASIS-30_Atropos_template_in_MNI152_2mm.nii.gz --TR 2 -s SUB_1024011
--subjects_dir fsdata --slice_times 0 17 1 18 2 19 3 20 4 21 5 22 6 23
7 24 8 25 9 26 10 27 11 28 12 29 13 30 14 31 15 32 16 -o .
```

This workflow takes resting timeseries and a Siemens dicom file corresponding to it and preprocesses it to produce timeseries coordinates or grayordinates.

This workflow also requires 2mm subcortical atlas and templates that are available from:

<http://mindboggle.info/data.html>

specifically the 2mm versions of:

- Joint Fusion Atlas
- MNI template

```
from __future__ import division, unicode_literals
from builtins import open, range, str

import os

from nipype.interfaces.base import CommandLine
CommandLine.set_default_terminal_output('allatonce')

from dicom import read_file
```

```

from nipy.interfaces import (spm, fsl, Function, ants, freesurfer)
from nipy.interfaces.c3 import C3dAffineTool

fsl.FSLCommand.set_default_output_type('NIFTI')

from nipy import Workflow, Node, MapNode
from nipy.interfaces import matlab as mlab

mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodisplay")
# If SPM is not in your MATLAB path you should add it here
# mlab.MatlabCommand.set_default_paths('/software/matlab/spm12')

from nipy.algorithms.rapidart import ArtifactDetect
from nipy.algorithms.misc import TSNR
from nipy.interfaces.utility import Rename, Merge, IdentityInterface
from nipy.utils.filemanip import filename_to_list
from nipy.interfaces.io import DataSink, FreeSurferSource

import numpy as np
import scipy as sp
import nibabel as nb

imports = ['import os',
          'import nibabel as nb',
          'import numpy as np',
          'import scipy as sp',
          'from nipy.utils.filemanip import filename_to_list, list_to_filename,
→ split_filename',
          'from scipy.special import legendre'
          ]

def get_info(dicom_files):
    from dcmstack.extract import default_extractor
    """Given a Siemens dicom file return metadata

    Returns
    -----
    RepetitionTime
    Slice Acquisition Times
    Spacing between slices
    """
    meta = default_extractor(read_file(filename_to_list(dicom_files)[0],
                                     stop_before_pixels=True,
                                     force=True))
    return (meta['RepetitionTime'] / 1000., meta['CsaImage.MosaicRefAcqTimes'],
            meta['SpacingBetweenSlices'])

def median(in_files):
    """Computes an average of the median of each realigned timeseries

    Parameters
    -----

    in_files: one or more realigned Nifti 4D time series

```


Returns

out_file: a 3D Nifti file

"""

```
import numpy as np
import nibabel as nb
from nipy.utils import NUMPY_MMAP
average = None
for idx, filename in enumerate(filename_to_list(in_files)):
    img = nb.load(filename, mmap=NUMPY_MMAP)
    data = np.median(img.get_data(), axis=3)
    if average is None:
        average = data
    else:
        average = average + data
median_img = nb.Nifti1Image(average / float(idx + 1), img.affine,
                             img.header)
filename = os.path.join(os.getcwd(), 'median.nii.gz')
median_img.to_filename(filename)
return filename
```

```
def bandpass_filter(files, lowpass_freq, highpass_freq, fs):
```

"""Bandpass filter the input files

Parameters

files: list of 4d nifti files

lowpass_freq: cutoff frequency for the low pass filter (in Hz)

highpass_freq: cutoff frequency for the high pass filter (in Hz)

fs: sampling rate (in Hz)

"""

```
from nipy.utils.filemanip import split_filename, list_to_filename
import numpy as np
import nibabel as nb
from nipy.utils import NUMPY_MMAP
out_files = []
for filename in filename_to_list(files):
    path, name, ext = split_filename(filename)
    out_file = os.path.join(os.getcwd(), name + '_bp' + ext)
    img = nb.load(filename, mmap=NUMPY_MMAP)
    timepoints = img.shape[-1]
    F = np.zeros((timepoints))
    lowidx = int(timepoints / 2) + 1
    if lowpass_freq > 0:
        lowidx = np.round(lowpass_freq / fs * timepoints)
    highidx = 0
    if highpass_freq > 0:
        highidx = np.round(highpass_freq / fs * timepoints)
    F[highidx:lowidx] = 1
    F = ((F + F[::-1]) > 0).astype(int)
    data = img.get_data()
    if np.all(F == 1):
        filtered_data = data
    else:
        filtered_data = np.real(np.fft.ifftn(np.fft.fftn(data) * F))
```

```
img_out = nb.Nifti1Image(filtered_data, img.affine, img.header)
img_out.to_filename(out_file)
out_files.append(out_file)
return list_to_filename(out_files)

def motion_regressors(motion_params, order=0, derivatives=1):
    """Compute motion regressors upto given order and derivative

    motion + d(motion)/dt + d2(motion)/dt2 (linear + quadratic)
    """
    import numpy as np
    out_files = []
    for idx, filename in enumerate(filename_to_list(motion_params)):
        params = np.genfromtxt(filename)
        out_params = params
        for d in range(1, derivatives + 1):
            cparams = np.vstack((np.repeat(params[0, :][None, :], d, axis=0),
                                params))
            out_params = np.hstack((out_params, np.diff(cparams, d, axis=0)))
        out_params2 = out_params
        for i in range(2, order + 1):
            out_params2 = np.hstack((out_params2, np.power(out_params, i)))
        filename = os.path.join(os.getcwd(), "motion_regressor%02d.txt" % idx)
        np.savetxt(filename, out_params2, fmt=b"%10f")
        out_files.append(filename)
    return out_files

def build_filter1(motion_params, comp_norm, outliers, detrend_poly=None):
    """Builds a regressor set comprising motion parameters, composite norm and
    outliers

    The outliers are added as a single time point column for each outlier

    Parameters
    -----

    motion_params: a text file containing motion parameters and its derivatives
    comp_norm: a text file containing the composite norm
    outliers: a text file containing 0-based outlier indices
    detrend_poly: number of polynomials to add to detrend

    Returns
    -----

    components_file: a text file containing all the regressors
    """
    import numpy as np
    import nibabel as nb
    from scipy.special import legendre
    out_files = []
    for idx, filename in enumerate(filename_to_list(motion_params)):
        params = np.genfromtxt(filename)
        norm_val = np.genfromtxt(filename_to_list(comp_norm)[idx])
        out_params = np.hstack((params, norm_val[:, None]))
        try:
            outlier_val = np.genfromtxt(filename_to_list(outliers)[idx])
```

```

except IOError:
    outlier_val = np.empty((0))
    for index in np.atleast_1d(outlier_val):
        outlier_vector = np.zeros((out_params.shape[0], 1))
        outlier_vector[index] = 1
        out_params = np.hstack((out_params, outlier_vector))
    if detrend_poly:
        timepoints = out_params.shape[0]
        X = np.empty((timepoints, 0))
        for i in range(detrend_poly):
            X = np.hstack((X, legendre(
                i + 1)(np.linspace(-1, 1, timepoints))[:, None]))
        out_params = np.hstack((out_params, X))
    filename = os.path.join(os.getcwd(), "filter_regressor%02d.txt" % idx)
    np.savetxt(filename, out_params, fmt=b"%10f")
    out_files.append(filename)
return out_files

def extract_noise_components(realigned_file, mask_file, num_components=5,
                           extra_regressors=None):
    """Derive components most reflective of physiological noise

    Parameters
    -----
    realigned_file: a 4D Nifti file containing realigned volumes
    mask_file: a 3D Nifti file containing white matter + ventricular masks
    num_components: number of components to use for noise decomposition
    extra_regressors: additional regressors to add

    Returns
    -----
    components_file: a text file containing the noise components
    """
    from scipy.linalg.decomp_svd import svd
    import numpy as np
    import nibabel as nb
    from nipy.utils import NUMPY_MMAP
    import os
    imgseries = nb.load(realigned_file, mmap=NUMPY_MMAP)
    components = None
    for filename in filename_to_list(mask_file):
        mask = nb.load(filename, mmap=NUMPY_MMAP).get_data()
        if len(np.nonzero(mask > 0)[0]) == 0:
            continue
        voxel_timecourses = imgseries.get_data()[mask > 0]
        voxel_timecourses[np.isnan(np.sum(voxel_timecourses, axis=1)), :] = 0
        # remove mean and normalize by variance
        # voxel_timecourses.shape == [nvoxels, time]
        X = voxel_timecourses.T
        stdX = np.std(X, axis=0)
        stdX[stdX == 0] = 1.
        stdX[np.isnan(stdX)] = 1.
        stdX[np.isinf(stdX)] = 1.
        X = (X - np.mean(X, axis=0)) / stdX
        u, _, _ = svd(X, full_matrices=False)
        if components is None:
            components = u[:, :num_components]

```

```

        else:
            components = np.hstack((components, u[:, :num_components]))
    if extra_regressors:
        regressors = np.genfromtxt(extra_regressors)
        components = np.hstack((components, regressors))
    components_file = os.path.join(os.getcwd(), 'noise_components.txt')
    np.savetxt(components_file, components, fmt=b"%10f")
    return components_file

def rename(in_files, suffix=None):
    from nipy.utils.filemanip import (filename_to_list, split_filename,
                                       list_to_filename)

    out_files = []
    for idx, filename in enumerate(filename_to_list(in_files)):
        _, name, ext = split_filename(filename)
        if suffix is None:
            out_files.append(name + ('_%03d' % idx) + ext)
        else:
            out_files.append(name + suffix + ext)
    return list_to_filename(out_files)

def get_aparc_aseg(files):
    """Return the aparc+aseg.mgz file"""
    for name in files:
        if 'aparc+aseg.mgz' in name:
            return name
    raise ValueError('aparc+aseg.mgz not found')

def extract_subrois(timeseries_file, label_file, indices):
    """Extract voxel time courses for each subcortical roi index

    Parameters
    -----

    timeseries_file: a 4D Nifti file
    label_file: a 3D file containing rois in the same space/size of the 4D file
    indices: a list of indices for ROIs to extract.

    Returns
    -----

    out_file: a text file containing time courses for each voxel of each roi
    The first four columns are: freesurfer index, i, j, k positions in the
    label file
    """
    from nipy.utils.filemanip import split_filename
    import nibabel as nb
    from nipy.utils import NUMPY_MMAP
    import os
    img = nb.load(timeseries_file, mmap=NUMPY_MMAP)
    data = img.get_data()
    roiimg = nb.load(label_file, mmap=NUMPY_MMAP)
    rois = roiimg.get_data()
    prefix = split_filename(timeseries_file)[1]
    out_ts_file = os.path.join(os.getcwd(), '%s_subcortical_ts.txt' % prefix)
    with open(out_ts_file, 'wt') as fp:

```

```

        for fsindex in indices:
            ijk = np.nonzero(rois == fsindex)
            ts = data[ijk]
            for i0, row in enumerate(ts):
                fp.write('%d,%d,%d,%d,' % (fsindex, ijk[0][i0],
                                           ijk[1][i0], ijk[2][i0]) +
                        ','.join(['%.10f' % val for val in row]) + '\n')
    return out_ts_file

def combine_hemi(left, right):
    """Combine left and right hemisphere time series into a single text file
    """
    import os
    import numpy as np
    from nipy.utils import NUMPY_MMAP
    lh_data = nb.load(left, mmap=NUMPY_MMAP).get_data()
    rh_data = nb.load(right, mmap=NUMPY_MMAP).get_data()

    indices = np.vstack((1000000 + np.arange(0, lh_data.shape[0])[:, None],
                        2000000 + np.arange(0, rh_data.shape[0])[:, None]))
    all_data = np.hstack((indices, np.vstack((lh_data.squeeze(),
                                             rh_data.squeeze()))))
    filename = left.split('.')[1] + '_combined.txt'
    np.savetxt(filename, all_data,
               fmt=','.join(['%d'] + ['%.10f'] * (all_data.shape[1] - 1)))
    return os.path.abspath(filename)

def create_reg_workflow(name='registration'):
    """Create a FEAT preprocessing workflow together with freesurfer

    Parameters
    -----

        name : name of workflow (default: 'registration')

    Inputs::

        inputspec.source_files : files (filename or list of filenames to register)
        inputspec.mean_image : reference image to use
        inputspec.anatomical_image : anatomical image to coregister to
        inputspec.target_image : registration target

    Outputs::

        outputspec.func2anat_transform : FLIRT transform
        outputspec.anat2target_transform : FLIRT+FNIRT transform
        outputspec.transformed_files : transformed files in target space
        outputspec.transformed_mean : mean image in target space
    """

    register = Workflow(name=name)

    inputnode = Node(interface=IdentityInterface(fields=['source_files',
                                                         'mean_image',
                                                         'subject_id',
                                                         'subjects_dir',

```

```

        name='inputspec')

        'target_image'])),

outputnode = Node(interface=IdentityInterface(fields=[
    'func2anat_transform',
    'out_reg_file',
    'anat2target_transform',
    'transforms',
    'transformed_mean',
    'segmentation_files',
    'anat2target',
    'aparc'
]),

        name='outputspec')

# Get the subject's freesurfer source directory
fssource = Node(FreeSurferSource(),
                name='fssource')
fssource.run_without_submitting = True
register.connect(inputnode, 'subject_id', fssource, 'subject_id')
register.connect(inputnode, 'subjects_dir', fssource, 'subjects_dir')

convert = Node(freesurfer.MRIConvert(out_type='nii'),
               name="convert")
register.connect(fssource, 'T1', convert, 'in_file')

# Coregister the median to the surface
bbregister = Node(freesurfer.BBRegister(),
                  name='bbregister')
bbregister.inputs.init = 'fsl'
bbregister.inputs.contrast_type = 't2'
bbregister.inputs.out_fsl_file = True
bbregister.inputs.epi_mask = True
register.connect(inputnode, 'subject_id', bbregister, 'subject_id')
register.connect(inputnode, 'mean_image', bbregister, 'source_file')
register.connect(inputnode, 'subjects_dir', bbregister, 'subjects_dir')

```

Estimate the tissue classes from the anatomical image. But use spm's segment as FSL appears to be breaking.

```

stripper = Node(fsl.BET(), name='stripper')
register.connect(convert, 'out_file', stripper, 'in_file')
fast = Node(fsl.FAST(), name='fast')
register.connect(stripper, 'out_file', fast, 'in_files')

```

Binarize the segmentation

```

binarize = MapNode(fsl.ImageMaths(op_string='-nan -thr 0.9 -ero -bin'),
                   iterfield=['in_file'],
                   name='binarize')
register.connect(fast, 'partial_volume_files', binarize, 'in_file')

```

Apply inverse transform to take segmentations to functional space

```

applyxfm = MapNode(freesurfer.ApplyVolTransform(inverse=True,
                                                  interp='nearest'),
                   iterfield=['target_file'],
                   name='inverse_transform')
register.connect(inputnode, 'subjects_dir', applyxfm, 'subjects_dir')
register.connect(bbregister, 'out_reg_file', applyxfm, 'reg_file')
register.connect(binarize, 'out_file', applyxfm, 'target_file')
register.connect(inputnode, 'mean_image', applyxfm, 'source_file')

```

Apply inverse transform to aparc file

```

aparcxfm = Node(freesurfer.ApplyVolTransform(inverse=True,
                                             interp='nearest'),
               name='aparc_inverse_transform')
register.connect(inputnode, 'subjects_dir', aparcxfm, 'subjects_dir')
register.connect(bbregister, 'out_reg_file', aparcxfm, 'reg_file')
register.connect(fssource, ('aparc_aseg', get_aparc_aseg),
               aparcxfm, 'target_file')
register.connect(inputnode, 'mean_image', aparcxfm, 'source_file')

```

Convert the BBRegister transformation to ANTS ITK format

```

convert2itk = Node(C3dAffineTool(), name='convert2itk')
convert2itk.inputs.fsl2ras = True
convert2itk.inputs.itk_transform = True
register.connect(bbregister, 'out_fsl_file', convert2itk, 'transform_file')
register.connect(inputnode, 'mean_image', convert2itk, 'source_file')
register.connect(stripper, 'out_file', convert2itk, 'reference_file')

```

Compute registration between the subject's structural and MNI template This is currently set to perform a very quick registration. However, the registration can be made significantly more accurate for cortical structures by increasing the number of iterations All parameters are set using the example from: [#https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh](https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh)

```

reg = Node(ants.Registration(), name='antsRegister')
reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = ['Rigid', 'Affine', 'SyN']
reg.inputs.transform_parameters = [(0.1,), (0.1,), (0.2, 3.0, 0.0)]
reg.inputs.number_of_iterations = [[10000, 11110, 11110]] * 2 + [[100, 30, 20]]
reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = True
reg.inputs.initial_moving_transform_com = True
reg.inputs.metric = ['Mattes'] * 2 + [['Mattes', 'CC']]
reg.inputs.metric_weight = [1] * 2 + [[0.5, 0.5]]
reg.inputs.radius_or_number_of_bins = [32] * 2 + [[32, 4]]
reg.inputs.sampling_strategy = ['Regular'] * 2 + [[None, None]]
reg.inputs.sampling_percentage = [0.3] * 2 + [[None, None]]
reg.inputs.convergence_threshold = [1.e-8] * 2 + [-0.01]
reg.inputs.convergence_window_size = [20] * 2 + [5]
reg.inputs.smoothing_sigmas = [[4, 2, 1]] * 2 + [[1, 0.5, 0]]
reg.inputs.sigma_units = ['vox'] * 3
reg.inputs.shrink_factors = [[3, 2, 1]] * 2 + [[4, 2, 1]]
reg.inputs.use_estimate_learning_rate_once = [True] * 3
reg.inputs.use_histogram_matching = [False] * 2 + [True]
reg.inputs.winsorize_lower_quantile = 0.005
reg.inputs.winsorize_upper_quantile = 0.995
reg.inputs.float = True
reg.inputs.output_warped_image = 'output_warped_image.nii.gz'
reg.inputs.num_threads = 4
reg.plugin_args = {'qsub_args': '-l nodes=1:ppn=4'}
register.connect(stripper, 'out_file', reg, 'moving_image')
register.connect(inputnode, 'target_image', reg, 'fixed_image')

```

Concatenate the affine and ants transforms into a list

```

merge = Node(Merge(2), iterfield=['in2'], name='mergexfm')
register.connect(convert2itk, 'itk_transform', merge, 'in2')

```

```
register.connect(reg, 'composite_transform', merge, 'in1')
```

Transform the mean image. First to anatomical and then to target

```
warpmean = Node(ants.ApplyTransforms(), name='warpmean')
warpmean.inputs.input_image_type = 3
warpmean.inputs.interpolation = 'Linear'
warpmean.inputs.invert_transform_flags = [False, False]
warpmean.inputs.terminal_output = 'file'
warpmean.inputs.args = '--float'
warpmean.inputs.num_threads = 4

register.connect(inputnode, 'target_image', warpmean, 'reference_image')
register.connect(inputnode, 'mean_image', warpmean, 'input_image')
register.connect(merge, 'out', warpmean, 'transforms')
```

Assign all the output files

```
register.connect(reg, 'warped_image', outputnode, 'anat2target')
register.connect(warpmean, 'output_image', outputnode, 'transformed_mean')
register.connect(applyxfm, 'transformed_file',
                 outputnode, 'segmentation_files')
register.connect(aparcxfm, 'transformed_file',
                 outputnode, 'aparc')
register.connect(bbregister, 'out_fsl_file',
                 outputnode, 'func2anat_transform')
register.connect(bbregister, 'out_reg_file',
                 outputnode, 'out_reg_file')
register.connect(reg, 'composite_transform',
                 outputnode, 'anat2target_transform')
register.connect(merge, 'out', outputnode, 'transforms')

return register
```

Creates the main preprocessing workflow

```
def create_workflow(files,
                    target_file,
                    subject_id,
                    TR,
                    slice_times,
                    norm_threshold=1,
                    num_components=5,
                    vol_fwhm=None,
                    surf_fwhm=None,
                    lowpass_freq=-1,
                    highpass_freq=-1,
                    subjects_dir=None,
                    sink_directory=os.getcwd(),
                    target_subject=['fsaverage3', 'fsaverage4'],
                    name='resting'):

    wf = Workflow(name=name)

    # Rename files in case they are named identically
    name_unique = MapNode(Rename(format_string='rest_%(run)02d',
                                  iterfield=['in_file', 'run'],
                                  name='rename'))
    name_unique.inputs.keep_ext = True
    name_unique.inputs.run = list(range(1, len(files) + 1))
```



```

name_unique.inputs.in_file = files

realign = Node(interface=spm.Realign(), name="realign")
realign.inputs.jobtype = 'estwrite'

num_slices = len(slice_times)
slice_timing = Node(interface=spm.SliceTiming(), name="slice_timing")
slice_timing.inputs.num_slices = num_slices
slice_timing.inputs.time_repetition = TR
slice_timing.inputs.time_acquisition = TR - TR / float(num_slices)
slice_timing.inputs.slice_order = (np.argsort(slice_times) + 1).tolist()
slice_timing.inputs.ref_slice = int(num_slices / 2)

# Compute TSNR on realigned data regressing polynomials upto order 2
tsnr = MapNode(TSNR(regress_poly=2), iterfield=['in_file'], name='tsnr')
wf.connect(slice_timing, 'timecorrected_files', tsnr, 'in_file')

# Compute the median image across runs
calc_median = Node(Function(input_names=['in_files'],
                             output_names=['median_file'],
                             function=median,
                             imports=imports),
                    name='median')
wf.connect(tsnr, 'detrended_file', calc_median, 'in_files')

```

Segment and Register

```

registration = create_reg_workflow(name='registration')
wf.connect(calc_median, 'median_file', registration, 'inputspec.mean_image')
registration.inputs.inputspec.subject_id = subject_id
registration.inputs.inputspec.subjects_dir = subjects_dir
registration.inputs.inputspec.target_image = target_file

```

Use `nipy.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```

art = Node(interface=ArtifactDetect(), name="art")
art.inputs.use_differences = [True, True]
art.inputs.use_norm = True
art.inputs.norm_threshold = norm_threshold
art.inputs.zintensity_threshold = 9
art.inputs.mask_type = 'spm_global'
art.inputs.parameter_source = 'SPM'

```

Here we are connecting all the nodes together. Notice that we add the merge node only if you choose to use 4D. Also `get_vox_dims` function is passed along the input volume of normalise to set the optimal voxel sizes.

```

wf.connect([(name_unique, realign, [('out_file', 'in_files')]),
           (realign, slice_timing, [('realigned_files', 'in_files')]),
           (slice_timing, art, [('timecorrected_files', 'realigned_files')]),
           (realign, art, [('realignment_parameters', 'realignment_parameters
→'])),
           ])

def selectindex(files, idx):
    import numpy as np
    from nipy.utils.filemanip import filename_to_list, list_to_filename
    return list_to_filename(np.array(filename_to_list(files))[idx].tolist())

```

```

mask = Node(fsl.BET(), name='getmask')
mask.inputs.mask = True
wf.connect(calc_median, 'median_file', mask, 'in_file')
# get segmentation in normalized functional space

def merge_files(in1, in2):
    out_files = filename_to_list(in1)
    out_files.extend(filename_to_list(in2))
    return out_files

# filter some noise

# Compute motion regressors
motreg = Node(Function(input_names=['motion_params', 'order',
                                   'derivatives'],
                       output_names=['out_files'],
                       function=motion_regressors,
                       imports=imports),
               name='getmotionregress')
wf.connect(realign, 'realignment_parameters', motreg, 'motion_params')

# Create a filter to remove motion and art confounds
createfilter1 = Node(Function(input_names=['motion_params', 'comp_norm',
                                           'outliers', 'detrend_poly'],
                              output_names=['out_files'],
                              function=build_filter1,
                              imports=imports),
                     name='makemotionbasedfilter')
createfilter1.inputs.detrend_poly = 2
wf.connect(motreg, 'out_files', createfilter1, 'motion_params')
wf.connect(art, 'norm_files', createfilter1, 'comp_norm')
wf.connect(art, 'outlier_files', createfilter1, 'outliers')

filter1 = MapNode(fsl.GLM(out_f_name='F_mcart.nii',
                          out_pf_name='pF_mcart.nii',
                          demean=True),
                  iterfield=['in_file', 'design', 'out_res_name'],
                  name='filtermotion')

wf.connect(slice_timing, 'timecorrected_files', filter1, 'in_file')
wf.connect(slice_timing, ('timecorrected_files', rename, '_filtermotart'),
           filter1, 'out_res_name')
wf.connect(createfilter1, 'out_files', filter1, 'design')

createfilter2 = MapNode(Function(input_names=['realigned_file', 'mask_file',
                                              'num_components',
                                              'extra_regressors'],
                              output_names=['out_files'],
                              function=extract_noise_components,
                              imports=imports),
                       iterfield=['realigned_file', 'extra_regressors'],
                       name='makecompcorrfilter')
createfilter2.inputs.num_components = num_components

wf.connect(createfilter1, 'out_files', createfilter2, 'extra_regressors')
wf.connect(filter1, 'out_res', createfilter2, 'realigned_file')
wf.connect(registration, ('outputspec.segmentation_files', selectindex, [0, 2]),
           createfilter2, 'mask_file')

```

```

filter2 = MapNode(fsl.GLM(out_f_name='F.nii',
                          out_pf_name='pF.nii',
                          demean=True),
                  iterfield=['in_file', 'design', 'out_res_name'],
                  name='filter_noise_nosmooth')
wf.connect(filter1, 'out_res', filter2, 'in_file')
wf.connect(filter1, ('out_res', rename, '_cleaned'),
           filter2, 'out_res_name')
wf.connect(createfilter2, 'out_files', filter2, 'design')
wf.connect(mask, 'mask_file', filter2, 'mask')

bandpass = Node(Function(input_names=['files', 'lowpass_freq',
                                     'highpass_freq', 'fs'],
                        output_names=['out_files'],
                        function=bandpass_filter,
                        imports=imports),
                name='bandpass_unsmooth')
bandpass.inputs.fs = 1. / TR
bandpass.inputs.highpass_freq = highpass_freq
bandpass.inputs.lowpass_freq = lowpass_freq
wf.connect(filter2, 'out_res', bandpass, 'files')

```

Smooth the functional data using `nipy.interfaces.spm.Smooth`.

```

smooth = Node(interface=spm.Smooth(), name="smooth")
smooth.inputs.fwhm = vol_fwhm

wf.connect(bandpass, 'out_files', smooth, 'in_files')

collector = Node(Merge(2), name='collect_streams')
wf.connect(smooth, 'smoothed_files', collector, 'in1')
wf.connect(bandpass, 'out_files', collector, 'in2')

```

Transform the remaining images. First to anatomical and then to target

```

warpall = MapNode(ants.ApplyTransforms(), iterfield=['input_image'],
                  name='warpall')
warpall.inputs.input_image_type = 3
warpall.inputs.interpolation = 'Linear'
warpall.inputs.invert_transform_flags = [False, False]
warpall.inputs.terminal_output = 'file'
warpall.inputs.reference_image = target_file
warpall.inputs.args = '--float'
warpall.inputs.num_threads = 1

# transform to target
wf.connect(collector, 'out', warpall, 'input_image')
wf.connect(registration, 'outputspec.transforms', warpall, 'transforms')

mask_target = Node(fsl.ImageMaths(op_string='-bin'), name='target_mask')

wf.connect(registration, 'outputspec.anat2target', mask_target, 'in_file')

maskts = MapNode(fsl.ApplyMask(), iterfield=['in_file'], name='ts_masker')
wf.connect(warpall, 'output_image', maskts, 'in_file')
wf.connect(mask_target, 'out_file', maskts, 'mask_file')

# map to surface

```

```

# extract aparc+aseg ROIs
# extract subcortical ROIs
# extract target space ROIs
# combine subcortical and cortical rois into a single cifti file

#####
# Convert aparc to subject functional space

# Sample the average time series in aparc ROIs
sampleaparc = MapNode(freesurfer.SegStats(default_color_table=True),
                      iterfield=['in_file', 'summary_file',
                                'avgwf_txt_file'],
                      name='aparc_ts')
sampleaparc.inputs.segment_id = ([8] + list(range(10, 14)) + [17, 18, 26, 47] +
                                list(range(49, 55)) + [58] + list(range(1001, 1036))) +
                                list(range(2001, 2036)))

wf.connect(registration, 'outputspec.aparc',
           sampleaparc, 'segmentation_file')
wf.connect(collector, 'out', sampleaparc, 'in_file')

def get_names(files, suffix):
    """Generate appropriate names for output files
    """
    from nipy.utils.filemanip import (split_filename, filename_to_list,
                                     list_to_filename)

    out_names = []
    for filename in files:
        _, name, _ = split_filename(filename)
        out_names.append(name + suffix)
    return list_to_filename(out_names)

wf.connect(collector, ('out', get_names, '_avgwf.txt'),
           sampleaparc, 'avgwf_txt_file')
wf.connect(collector, ('out', get_names, '_summary.stats'),
           sampleaparc, 'summary_file')

# Sample the time series onto the surface of the target surface. Performs
# sampling into left and right hemisphere
target = Node(IdentityInterface(fields=['target_subject']), name='target')
target.iterables = ('target_subject', filename_to_list(target_subject))

samplerlh = MapNode(freesurfer.SampleToSurface(),
                    iterfield=['source_file'],
                    name='sampler_lh')
samplerlh.inputs.sampling_method = "average"
samplerlh.inputs.sampling_range = (0.1, 0.9, 0.1)
samplerlh.inputs.sampling_units = "frac"
samplerlh.inputs.interp_method = "trilinear"
samplerlh.inputs.smooth_surf = surf_fwhm
# samplerlh.inputs.cortex_mask = True
samplerlh.inputs.out_type = 'niigz'
samplerlh.inputs.subjects_dir = subjects_dir

samplerlh = samplerlh.clone('sampler_rh')

samplerlh.inputs.hemi = 'lh'

```

```

wf.connect(collector, 'out', samplerlh, 'source_file')
wf.connect(registration, 'outputspec.out_reg_file', samplerlh, 'reg_file')
wf.connect(target, 'target_subject', samplerlh, 'target_subject')

samplerrh.set_input('hemi', 'rh')
wf.connect(collector, 'out', samplerrh, 'source_file')
wf.connect(registration, 'outputspec.out_reg_file', samplerrh, 'reg_file')
wf.connect(target, 'target_subject', samplerrh, 'target_subject')

# Combine left and right hemisphere to text file
combiner = MapNode(Function(input_names=['left', 'right'],
                           output_names=['out_file'],
                           function=combine_hemi,
                           imports=imports),
                   iterfield=['left', 'right'],
                   name="combiner")
wf.connect(samplerlh, 'out_file', combiner, 'left')
wf.connect(samplerrh, 'out_file', combiner, 'right')

# Sample the time series file for each subcortical roi
ts2txt = MapNode(Function(input_names=['timeseries_file', 'label_file',
                                       'indices'],
                           output_names=['out_file'],
                           function=extract_subrois,
                           imports=imports),
                  iterfield=['timeseries_file'],
                  name='getsubcortts')
ts2txt.inputs.indices = [8] + list(range(10, 14)) + [17, 18, 26, 47] + \
    list(range(49, 55)) + [58]
ts2txt.inputs.label_file = \
    os.path.abspath(('OASIS-TRT-20_jointfusion_DKT31_CMA_labels_in_MNI152_'
                    '2mm_v2.nii.gz'))
wf.connect(maskts, 'out_file', ts2txt, 'timeseries_file')

#####

substitutions = [('_target_subject_', ''),
                 ('_filtermotart_cleaned_bp_trans_masked', ''),
                 ('_filtermotart_cleaned_bp', '')
                ]
regex_subs = [('_ts_masker.*sar', '/smooth/'),
              ('_ts_masker.*ar', '/unsmooth/'),
              ('_combiner.*sar', '/smooth/'),
              ('_combiner.*ar', '/unsmooth/'),
              ('_aparc_ts.*sar', '/smooth/'),
              ('_aparc_ts.*ar', '/unsmooth/'),
              ('_getsubcortts.*sar', '/smooth/'),
              ('_getsubcortts.*ar', '/unsmooth/'),
              ('series/sar', 'series/smooth/'),
              ('series/ar', 'series/unsmooth/'),
              ('_inverse_transform./', ''),
              ]

# Save the relevant data into an output directory
datasink = Node(interface=DataSink(), name="datasink")
datasink.inputs.base_directory = sink_directory
datasink.inputs.container = subject_id
datasink.inputs.substitutions = substitutions
datasink.inputs.regexp_substitutions = regex_subs # (r'(/_.*(\d+))', r'/run\2')

```

```

wf.connect(realign, 'realignment_parameters', datasink, 'resting.qa.motion')
wf.connect(art, 'norm_files', datasink, 'resting.qa.art.@norm')
wf.connect(art, 'intensity_files', datasink, 'resting.qa.art.@intensity')
wf.connect(art, 'outlier_files', datasink, 'resting.qa.art.@outlier_files')
wf.connect(registration, 'outputspec.segmentation_files', datasink, 'resting.mask_
↪files')
wf.connect(registration, 'outputspec.anat2target', datasink, 'resting.qa.ants')
wf.connect(mask, 'mask_file', datasink, 'resting.mask_files.@brainmask')
wf.connect(mask_target, 'out_file', datasink, 'resting.mask_files.target')
wf.connect(filter1, 'out_f', datasink, 'resting.qa.compmaps.@mc_F')
wf.connect(filter1, 'out_pf', datasink, 'resting.qa.compmaps.@mc_pF')
wf.connect(filter2, 'out_f', datasink, 'resting.qa.compmaps')
wf.connect(filter2, 'out_pf', datasink, 'resting.qa.compmaps.@p')
wf.connect(bandpass, 'out_files', datasink, 'resting.timeseries.@bandpassed')
wf.connect(smooth, 'smoothed_files', datasink, 'resting.timeseries.@smoothed')
wf.connect(createfilter1, 'out_files',
            datasink, 'resting.regress.@regressors')
wf.connect(createfilter2, 'out_files',
            datasink, 'resting.regress.@compcorr')
wf.connect(maskts, 'out_file', datasink, 'resting.timeseries.target')
wf.connect(sampleaparc, 'summary_file',
            datasink, 'resting.parcellations.aparc')
wf.connect(sampleaparc, 'avgwf_txt_file',
            datasink, 'resting.parcellations.aparc.@avgwf')
wf.connect(ts2txt, 'out_file',
            datasink, 'resting.parcellations.grayo.@subcortical')

datasink2 = Node(interface=DataSink(), name="datasink2")
datasink2.inputs.base_directory = sink_directory
datasink2.inputs.container = subject_id
datasink2.inputs.substitutions = substitutions
datasink2.inputs.regex_substitutions = regex_subs # (r'(/_.*(\d+))', r'/run\2')
wf.connect(combiner, 'out_file',
            datasink2, 'resting.parcellations.grayo.@surface')

return wf

```

Creates the full workflow including getting information from dicom files

```

def create_resting_workflow(args, name=None):
    TR = args.TR
    slice_times = args.slice_times
    if args.dicom_file:
        TR, slice_times, slice_thickness = get_info(args.dicom_file)
        slice_times = (np.array(slice_times) / 1000.).tolist()
    if name is None:
        name = 'resting_' + args.subject_id
    kwargs = dict(files=[os.path.abspath(filename) for filename in args.files],
                  target_file=os.path.abspath(args.target_file),
                  subject_id=args.subject_id,
                  TR=TR,
                  slice_times=slice_times,
                  vol_fwhm=args.vol_fwhm,
                  surf_fwhm=args.surf_fwhm,
                  norm_threshold=2.,
                  subjects_dir=os.path.abspath(args.fsd_dir),
                  target_subject=args.target_surfs,
                  lowpass_freq=args.lowpass_freq,
                  highpass_freq=args.highpass_freq,

```

```

        sink_directory=os.path.abspath(args.sink),
        name=name)
wf = create_workflow(**kwargs)
return wf

if __name__ == "__main__":
    from argparse import ArgumentParser, RawTextHelpFormatter
    defstr = ' (default %(default)s)'
    parser = ArgumentParser(description=__doc__,
                              formatter_class=RawTextHelpFormatter)
    parser.add_argument("-d", "--dicom_file", dest="dicom_file",
                        help="an example dicom file from the resting series")
    parser.add_argument("-f", "--files", dest="files", nargs="+",
                        help="4d nifti files for resting state",
                        required=True)
    parser.add_argument("-t", "--target", dest="target_file",
                        help=("Target in MNI space. Best to use the MindBoggle "
                              "template - "
                              "OASIS-30_Atropos_template_in_MNI152_2mm.nii.gz"),
                        required=True)
    parser.add_argument("-s", "--subject_id", dest="subject_id",
                        help="FreeSurfer subject id", required=True)
    parser.add_argument("--subjects_dir", dest="fsdir",
                        help="FreeSurfer subject directory", required=True)
    parser.add_argument("--target_surfaces", dest="target_surfs", nargs="+",
                        default=['fsaverage5'],
                        help="FreeSurfer target surfaces" + defstr)
    parser.add_argument("--TR", dest="TR", default=None, type=float,
                        help="TR if dicom not provided in seconds")
    parser.add_argument("--slice_times", dest="slice_times", nargs="+",
                        type=float, help="Slice onset times in seconds")
    parser.add_argument("--vol_fwhm", default=6., dest='vol_fwhm',
                        type=float, help="Spatial FWHM" + defstr)
    parser.add_argument("--surf_fwhm", default=15., dest='surf_fwhm',
                        type=float, help="Spatial FWHM" + defstr)
    parser.add_argument("-l", "--lowpass_freq", dest="lowpass_freq",
                        default=0.1, type=float,
                        help="Low pass frequency (Hz)" + defstr)
    parser.add_argument("-u", "--highpass_freq", dest="highpass_freq",
                        default=0.01, type=float,
                        help="High pass frequency (Hz)" + defstr)
    parser.add_argument("-o", "--output_dir", dest="sink",
                        help="Output directory base", required=True)
    parser.add_argument("-w", "--work_dir", dest="work_dir",
                        help="Output directory base")
    parser.add_argument("-p", "--plugin", dest="plugin",
                        default='Linear',
                        help="Plugin to use")
    parser.add_argument("--plugin_args", dest="plugin_args",
                        help="Plugin arguments")
    args = parser.parse_args()

    wf = create_resting_workflow(args)

    if args.work_dir:
        work_dir = os.path.abspath(args.work_dir)
    else:
        work_dir = os.getcwd()

```

```
wf.base_dir = work_dir
if args.plugin_args:
    wf.run(args.plugin, plugin_args=eval(args.plugin_args))
else:
    wf.run(args.plugin)
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

rsfMRI: ANTS, FS, FSL, NiPy, aCompCor

A preprocessing workflow for Siemens resting state data.

This workflow makes use of:

- ANTS
- FreeSurfer
- FSL
- NiPy
- CompCor

For example:

```
python rsfmri_preprocessing.py -d /data/12345-34-1.dcm -f /data/Resting.nii
-s subj001 -o output -p PBS --plugin_args "dict(qsub_args='-q many')"
```

or

```
python rsfmri_vol_surface_preprocessing.py -f SUB_1024011/E?/func/rest.nii
-t OASIS-30_Atropos_template_in_MNI152_2mm.nii.gz --TR 2 -s SUB_1024011
--subjects_dir fsdata --slice_times 0 17 1 18 2 19 3 20 4 21 5 22 6 23
7 24 8 25 9 26 10 27 11 28 12 29 13 30 14 31 15 32 16 -o .
```

This workflow takes resting timeseries and a Siemens dicom file corresponding to it and preprocesses it to produce timeseries coordinates or grayordinates.

For non-Siemens dicoms, provide slice times instead, since the dicom extractor is not guaranteed to work.

This workflow also requires 2mm subcortical atlas and templates that are available from:

<http://mindboggle.info/data.html>

specifically the 2mm versions of:

- Joint Fusion Atlas
- MNI template

```
from __future__ import division, unicode_literals
from builtins import open, range, str

import os

from nipy.interfaces.base import CommandLine
CommandLine.set_default_terminal_output('allatonce')

# https://github.com/moloney/dcmstack
```

```

from dcmstack.extract import default_extractor
# pip install pydicom
from dicom import read_file

from nipy.interfaces import (fsl, Function, ants, freesurfer, nipy)
from nipy.interfaces.c3 import C3dAffineTool

fsl.FSLCommand.set_default_output_type('NIFTI_GZ')

from nipy import Workflow, Node, MapNode

from nipy.algorithms.rapidart import ArtifactDetect
from nipy.algorithms.misc import TSNR
from nipy.algorithms.confounds import ACompCor
from nipy.interfaces.utility import Rename, Merge, IdentityInterface
from nipy.utils.filemanip import filename_to_list
from nipy.interfaces.io import DataSink, FreeSurferSource
import nipy.interfaces.freesurfer as fs

import numpy as np
import scipy as sp
import nibabel as nb
from nipy.utils.config import NUMPY_MMAP

imports = ['import os',
           'import nibabel as nb',
           'import numpy as np',
           'import scipy as sp',
           'from nipy.utils.filemanip import filename_to_list, list_to_filename,
→ split_filename',
           'from scipy.special import legendre'
          ]

def get_info(dicom_files):
    """Given a Siemens dicom file return metadata

    Returns
    -----
    RepetitionTime
    Slice Acquisition Times
    Spacing between slices
    """
    meta = default_extractor(read_file(filename_to_list(dicom_files)[0],
                                         stop_before_pixels=True,
                                         force=True))
    return (meta['RepetitionTime'] / 1000., meta['CsaImage.MosaicRefAcqTimes'],
            meta['SpacingBetweenSlices'])

def median(in_files):
    """Computes an average of the median of each realigned timeseries

    Parameters
    -----

    in_files: one or more realigned Nifti 4D time series

```

Returns

out_file: a 3D Nifti file

"""

average = **None**

for idx, filename **in** enumerate(filename_to_list(in_files)):

img = nb.load(filename, mmap=NUMPY_MMAP)

data = np.median(img.get_data(), axis=3)

if average **is** **None**:

average = data

else:

average = average + data

median_img = nb.Nifti1Image(average / float(idx + 1), img.affine,
img.header)

filename = os.path.join(os.getcwd(), 'median.nii.gz')

median_img.to_filename(filename)

return filename

def bandpass_filter(files, lowpass_freq, highpass_freq, fs):

"""Bandpass filter the input files

Parameters

files: list of 4d nifti files

lowpass_freq: cutoff frequency for the low pass filter (in Hz)

highpass_freq: cutoff frequency for the high pass filter (in Hz)

fs: sampling rate (in Hz)

"""

out_files = []

for filename **in** filename_to_list(files):

path, name, ext = split_filename(filename)

out_file = os.path.join(os.getcwd(), name + '_bp' + ext)

img = nb.load(filename, mmap=NUMPY_MMAP)

timepoints = img.shape[-1]

F = np.zeros((timepoints))

lowidx = int(timepoints / 2) + 1

if lowpass_freq > 0:

lowidx = np.round(float(lowpass_freq) / fs * timepoints)

highidx = 0

if highpass_freq > 0:

highidx = np.round(float(highpass_freq) / fs * timepoints)

F[highidx:lowidx] = 1

F = ((F + F[::-1]) > 0).astype(int)

data = img.get_data()

if np.all(F == 1):

filtered_data = data

else:

filtered_data = np.real(np.fft.ifftn(np.fft.fftn(data) * F))

img_out = nb.Nifti1Image(filtered_data, img.affine, img.header)

img_out.to_filename(out_file)

out_files.append(out_file)

return list_to_filename(out_files)

def motion_regressors(motion_params, order=0, derivatives=1):

"""Compute motion regressors upto given order and derivative

```

motion + d(motion)/dt + d2(motion)/dt2 (linear + quadratic)
"""
out_files = []
for idx, filename in enumerate(filename_to_list(motion_params)):
    params = np.genfromtxt(filename)
    out_params = params
    for d in range(1, derivatives + 1):
        cparams = np.vstack((np.repeat(params[0, :][None, :], d, axis=0),
                               params))
        out_params = np.hstack((out_params, np.diff(cparams, d, axis=0)))
    out_params2 = out_params
    for i in range(2, order + 1):
        out_params2 = np.hstack((out_params2, np.power(out_params, i)))
    filename = os.path.join(os.getcwd(), "motion_regressor%02d.txt" % idx)
    np.savetxt(filename, out_params2, fmt=b"%10f")
    out_files.append(filename)
return out_files

def build_filter1(motion_params, comp_norm, outliers, detrend_poly=None):
    """Builds a regressor set comprising motion parameters, composite norm and
    outliers

    The outliers are added as a single time point column for each outlier

    Parameters
    -----

    motion_params: a text file containing motion parameters and its derivatives
    comp_norm: a text file containing the composite norm
    outliers: a text file containing 0-based outlier indices
    detrend_poly: number of polynomials to add to detrend

    Returns
    -----

    components_file: a text file containing all the regressors
    """
    out_files = []
    for idx, filename in enumerate(filename_to_list(motion_params)):
        params = np.genfromtxt(filename)
        norm_val = np.genfromtxt(filename_to_list(comp_norm)[idx])
        out_params = np.hstack((params, norm_val[:, None]))
        try:
            outlier_val = np.genfromtxt(filename_to_list(outliers)[idx])
        except IOError:
            outlier_val = np.empty((0))
        for index in np.atleast_1d(outlier_val):
            outlier_vector = np.zeros((out_params.shape[0], 1))
            outlier_vector[index] = 1
            out_params = np.hstack((out_params, outlier_vector))
        if detrend_poly:
            timepoints = out_params.shape[0]
            X = np.empty((timepoints, 0))
            for i in range(detrend_poly):
                X = np.hstack((X, legendre(
                    i + 1)(np.linspace(-1, 1, timepoints))[:, None])))

```

```

        out_params = np.hstack((out_params, X))
        filename = os.path.join(os.getcwd(), "filter_regressor%02d.txt" % idx)
        np.savetxt(filename, out_params, fmt=b"%10f")
        out_files.append(filename)
    return out_files

def rename(in_files, suffix=None):
    from nipyype.utils.filemanip import (filename_to_list, split_filename,
                                         list_to_filename)

    out_files = []
    for idx, filename in enumerate(filename_to_list(in_files)):
        _, name, ext = split_filename(filename)
        if suffix is None:
            out_files.append(name + ('_%03d' % idx) + ext)
        else:
            out_files.append(name + suffix + ext)
    return list_to_filename(out_files)

def get_aparc_aseg(files):
    """Return the aparc+aseg.mgz file"""
    for name in files:
        if 'aparc+aseg.mgz' in name:
            return name
    raise ValueError('aparc+aseg.mgz not found')

def extract_subrois(timeseries_file, label_file, indices):
    """Extract voxel time courses for each subcortical roi index

    Parameters
    -----

    timeseries_file: a 4D Nifti file
    label_file: a 3D file containing rois in the same space/size of the 4D file
    indices: a list of indices for ROIs to extract.

    Returns
    -----

    out_file: a text file containing time courses for each voxel of each roi
    The first four columns are: freesurfer index, i, j, k positions in the
    label file
    """
    img = nb.load(timeseries_file, mmap=NUMPY_MMAP)
    data = img.get_data()
    roiimg = nb.load(label_file, mmap=NUMPY_MMAP)
    rois = roiimg.get_data()
    prefix = split_filename(timeseries_file)[1]
    out_ts_file = os.path.join(os.getcwd(), '%s_subcortical_ts.txt' % prefix)
    with open(out_ts_file, 'wt') as fp:
        for fsindex in indices:
            ijk = np.nonzero(rois == fsindex)
            ts = data[ijk]
            for i0, row in enumerate(ts):
                fp.write('%d,%d,%d,%d,' % (fsindex, ijk[0][i0],
                                           ijk[1][i0], ijk[2][i0]) +
                        ','.join(['%.10f' % val for val in row]) + '\n')
    return out_ts_file

```

```
def combine_hemi(left, right):
    """Combine left and right hemisphere time series into a single text file
    """
    lh_data = nb.load(left, mmap=NUMPY_MMAP).get_data()
    rh_data = nb.load(right, mmap=NUMPY_MMAP).get_data()

    indices = np.vstack((1000000 + np.arange(0, lh_data.shape[0])[:, None],
                        2000000 + np.arange(0, rh_data.shape[0])[:, None]))
    all_data = np.hstack((indices, np.vstack((lh_data.squeeze(),
                                             rh_data.squeeze()))))

    filename = left.split('.')[1] + '_combined.txt'
    np.savetxt(filename, all_data,
               fmt=','.join(['%d'] + ['%.10f'] * (all_data.shape[1] - 1)))
    return os.path.abspath(filename)

def create_reg_workflow(name='registration'):
    """Create a FEAT preprocessing workflow together with freesurfer

    Parameters
    -----

        name : name of workflow (default: 'registration')

    Inputs::

        inputspec.source_files : files (filename or list of filenames to register)
        inputspec.mean_image : reference image to use
        inputspec.anatomical_image : anatomical image to coregister to
        inputspec.target_image : registration target

    Outputs::

        outputspec.func2anat_transform : FLIRT transform
        outputspec.anat2target_transform : FLIRT+FNIRT transform
        outputspec.transformed_files : transformed files in target space
        outputspec.transformed_mean : mean image in target space
    """

    register = Workflow(name=name)

    inputnode = Node(interface=IdentityInterface(fields=['source_files',
                                                         'mean_image',
                                                         'subject_id',
                                                         'subjects_dir',
                                                         'target_image']),
                     name='inputspec')

    outputnode = Node(interface=IdentityInterface(fields=['func2anat_transform',
                                                         'out_reg_file',
                                                         'anat2target_transform',
                                                         'transforms',
                                                         'transformed_mean',
                                                         'segmentation_files',
                                                         'anat2target',
                                                         'aparc'],
```

```

        'min_cost_file'
    ]),

    name='outputspec')

# Get the subject's freesurfer source directory
fssource = Node(FreeSurferSource(),
    name='fssource')
fssource.run_without_submitting = True
register.connect(inputnode, 'subject_id', fssource, 'subject_id')
register.connect(inputnode, 'subjects_dir', fssource, 'subjects_dir')

convert = Node(freesurfer.MRIConvert(out_type='nii'),
    name="convert")
register.connect(fssource, 'T1', convert, 'in_file')

# Coregister the median to the surface
bbregister = Node(freesurfer.BBRegister(),
    name='bbregister')
bbregister.inputs.init = 'fsl'
bbregister.inputs.contrast_type = 't2'
bbregister.inputs.out_fsl_file = True
bbregister.inputs.epi_mask = True
register.connect(inputnode, 'subject_id', bbregister, 'subject_id')
register.connect(inputnode, 'mean_image', bbregister, 'source_file')
register.connect(inputnode, 'subjects_dir', bbregister, 'subjects_dir')

```

Estimate the tissue classes from the anatomical image. But use aparc+aseg's brain mask

```

binarize = Node(fs.Binarize(min=0.5, out_type="nii.gz", dilate=1), name="binarize_
↪aparc")
register.connect(fssource, ("aparc_aseg", get_aparc_aseg), binarize, "in_file")
stripper = Node(fsl.ApplyMask(), name='stripper')
register.connect(binarize, "binary_file", stripper, "mask_file")
register.connect(convert, 'out_file', stripper, 'in_file')

fast = Node(fsl.FAST(), name='fast')
register.connect(stripper, 'out_file', fast, 'in_files')

```

Binarize the segmentation

```

binarize = MapNode(fsl.ImageMaths(op_string='-nan -thr 0.9 -ero -bin'),
    iterfield=['in_file'],
    name='binarize')
register.connect(fast, 'partial_volume_files', binarize, 'in_file')

```

Apply inverse transform to take segmentations to functional space

```

applyxfm = MapNode(freesurfer.ApplyVolTransform(inverse=True,
    interp='nearest'),
    iterfield=['target_file'],
    name='inverse_transform')
register.connect(inputnode, 'subjects_dir', applyxfm, 'subjects_dir')
register.connect(bbregister, 'out_reg_file', applyxfm, 'reg_file')
register.connect(binarize, 'out_file', applyxfm, 'target_file')
register.connect(inputnode, 'mean_image', applyxfm, 'source_file')

```

Apply inverse transform to aparc file

```

aparcxfm = Node(freesurfer.ApplyVolTransform(inverse=True,
    interp='nearest'),

```

```

        name='aparc_inverse_transform')
register.connect(inputnode, 'subjects_dir', aparccfm, 'subjects_dir')
register.connect(bregister, 'out_reg_file', aparccfm, 'reg_file')
register.connect(fssource, ('aparc_aseg', get_aparc_aseg),
                 aparccfm, 'target_file')
register.connect(inputnode, 'mean_image', aparccfm, 'source_file')

```

Convert the BBRegister transformation to ANTS ITK format

```

convert2itk = Node(C3dAffineTool(), name='convert2itk')
convert2itk.inputs.fsl2ras = True
convert2itk.inputs.itk_transform = True
register.connect(bregister, 'out_fsl_file', convert2itk, 'transform_file')
register.connect(inputnode, 'mean_image', convert2itk, 'source_file')
register.connect(stripper, 'out_file', convert2itk, 'reference_file')

```

Compute registration between the subject's structural and MNI template This is currently set to perform a very quick registration. However, the registration can be made significantly more accurate for cortical structures by increasing the number of iterations All parameters are set using the example from: [#https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh](https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh)

```

reg = Node(ants.Registration(), name='antsRegister')
reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = ['Rigid', 'Affine', 'SyN']
reg.inputs.transform_parameters = [(0.1,), (0.1,), (0.2, 3.0, 0.0)]
reg.inputs.number_of_iterations = [[10000, 11110, 11110]] * 2 + [[100, 30, 20]]
reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = True
reg.inputs.initial_moving_transform_com = True
reg.inputs.metric = ['Mattes'] * 2 + [['Mattes', 'CC']]
reg.inputs.metric_weight = [1] * 2 + [[0.5, 0.5]]
reg.inputs.radius_or_number_of_bins = [32] * 2 + [[32, 4]]
reg.inputs.sampling_strategy = ['Regular'] * 2 + [[None, None]]
reg.inputs.sampling_percentage = [0.3] * 2 + [[None, None]]
reg.inputs.convergence_threshold = [1.e-8] * 2 + [-0.01]
reg.inputs.convergence_window_size = [20] * 2 + [5]
reg.inputs.smoothing_sigmas = [[4, 2, 1]] * 2 + [[1, 0.5, 0]]
reg.inputs.sigma_units = ['vox'] * 3
reg.inputs.shrink_factors = [[3, 2, 1]] * 2 + [[4, 2, 1]]
reg.inputs.use_estimate_learning_rate_once = [True] * 3
reg.inputs.use_histogram_matching = [False] * 2 + [True]
reg.inputs.winsorize_lower_quantile = 0.005
reg.inputs.winsorize_upper_quantile = 0.995
reg.inputs.float = True
reg.inputs.output_warped_image = 'output_warped_image.nii.gz'
reg.inputs.num_threads = 4
reg.plugin_args = {'sbatch_args': '-c%d' % 4}
register.connect(stripper, 'out_file', reg, 'moving_image')
register.connect(inputnode, 'target_image', reg, 'fixed_image')

```

Concatenate the affine and ants transforms into a list

```

merge = Node(Merge(2), iterfield=['in2'], name='mergexfm')
register.connect(convert2itk, 'itk_transform', merge, 'in2')
register.connect(reg, 'composite_transform', merge, 'in1')

```

Transform the mean image. First to anatomical and then to target


```

warpmean = Node(ants.ApplyTransforms(), name='warpmean')
warpmean.inputs.input_image_type = 3
warpmean.inputs.interpolation = 'Linear'
warpmean.inputs.invert_transform_flags = [False, False]
warpmean.inputs.terminal_output = 'file'
warpmean.inputs.args = '--float'
warpmean.inputs.num_threads = 4
warpmean.plugin_args = {'sbatch_args': '-c%d' % 4}

register.connect(inputnode, 'target_image', warpmean, 'reference_image')
register.connect(inputnode, 'mean_image', warpmean, 'input_image')
register.connect(merge, 'out', warpmean, 'transforms')

```

Assign all the output files

```

register.connect(reg, 'warped_image', outputnode, 'anat2target')
register.connect(warpmean, 'output_image', outputnode, 'transformed_mean')
register.connect(applyxfm, 'transformed_file',
                 outputnode, 'segmentation_files')
register.connect(aparcxfm, 'transformed_file',
                 outputnode, 'aparc')
register.connect(bbregister, 'out_fsl_file',
                 outputnode, 'func2anat_transform')
register.connect(bbregister, 'out_reg_file',
                 outputnode, 'out_reg_file')
register.connect(reg, 'composite_transform',
                 outputnode, 'anat2target_transform')
register.connect(merge, 'out', outputnode, 'transforms')
register.connect(bbregister, 'min_cost_file',
                 outputnode, 'min_cost_file')

return register

```

Creates the main preprocessing workflow

```

def create_workflow(files,
                    target_file,
                    subject_id,
                    TR,
                    slice_times,
                    norm_threshold=1,
                    num_components=5,
                    vol_fwhm=None,
                    surf_fwhm=None,
                    lowpass_freq=-1,
                    highpass_freq=-1,
                    subjects_dir=None,
                    sink_directory=os.getcwd(),
                    target_subject=['fsaverage3', 'fsaverage4'],
                    name='resting'):

    wf = Workflow(name=name)

    # Rename files in case they are named identically
    name_unique = MapNode(Rename(format_string='rest_%(run)02d'),
                          iterfield=['in_file', 'run'],
                          name='rename')
    name_unique.inputs.keep_ext = True
    name_unique.inputs.run = list(range(1, len(files) + 1))

```

```

name_unique.inputs.in_file = files

realign = Node(nipy.SpaceTimeRealigner(), name="spacetime_realign")
realign.inputs.slice_times = slice_times
realign.inputs.tr = TR
realign.inputs.slice_info = 2
realign.plugin_args = {'sbatch_args': '-c%d' % 4}

# Compute TSNR on realigned data regressing polynomials up to order 2
tsnr = MapNode(TSNR(regress_poly=2), iterfield=['in_file'], name='tsnr')
wf.connect(realign, "out_file", tsnr, "in_file")

# Compute the median image across runs
calc_median = Node(Function(input_names=['in_files'],
                             output_names=['median_file'],
                             function=median,
                             imports=imports),
                    name='median')
wf.connect(tsnr, 'detrended_file', calc_median, 'in_files')

```

Segment and Register

```

registration = create_reg_workflow(name='registration')
wf.connect(calc_median, 'median_file', registration, 'inputspec.mean_image')
registration.inputs.inputspec.subject_id = subject_id
registration.inputs.inputspec.subjects_dir = subjects_dir
registration.inputs.inputspec.target_image = target_file

```

Quantify TSNR in each freesurfer ROI

```

get_roi_tsnr = MapNode(fs.SegStats(default_color_table=True),
                       iterfield=['in_file'], name='get_aparc_tsnr')
get_roi_tsnr.inputs.avgwf_txt_file = True
wf.connect(tsnr, 'tsnr_file', get_roi_tsnr, 'in_file')
wf.connect(registration, 'outputspec.aparc', get_roi_tsnr, 'segmentation_file')

```

Use `nipy.algorithms.rapidart` to determine which of the images in the functional series are outliers based on deviations in intensity or movement.

```

art = Node(interface=ArtifactDetect(), name="art")
art.inputs.use_differences = [True, True]
art.inputs.use_norm = True
art.inputs.norm_threshold = norm_threshold
art.inputs.zintensity_threshold = 9
art.inputs.mask_type = 'spm_global'
art.inputs.parameter_source = 'NiPy'

```

Here we are connecting all the nodes together. Notice that we add the merge node only if you choose to use 4D. Also `get_vox_dims` function is passed along the input volume of normalise to set the optimal voxel sizes.

```

wf.connect([(name_unique, realign, [('out_file', 'in_file')]),
           (realign, art, [('out_file', 'realigned_files')]),
           (realign, art, [('par_file', 'realignment_parameters'])],
           ])

def selectindex(files, idx):
    import numpy as np
    from nipy.utils.filemanip import filename_to_list, list_to_filename
    return list_to_filename(np.array(filename_to_list(files))[idx].tolist())

```

```

mask = Node(fsl.BET(), name='getmask')
mask.inputs.mask = True
wf.connect(calc_median, 'median_file', mask, 'in_file')
# get segmentation in normalized functional space

def merge_files(in1, in2):
    out_files = filename_to_list(in1)
    out_files.extend(filename_to_list(in2))
    return out_files

# filter some noise

# Compute motion regressors
motreg = Node(Function(input_names=['motion_params', 'order',
                                   'derivatives'],
                       output_names=['out_files'],
                       function=motion_regressors,
                       imports=imports),
               name='getmotionregress')
wf.connect(realign, 'par_file', motreg, 'motion_params')

# Create a filter to remove motion and art confounds
createfilter1 = Node(Function(input_names=['motion_params', 'comp_norm',
                                           'outliers', 'detrend_poly'],
                              output_names=['out_files'],
                              function=build_filter1,
                              imports=imports),
                     name='makemotionbasedfilter')
createfilter1.inputs.detrend_poly = 2
wf.connect(motreg, 'out_files', createfilter1, 'motion_params')
wf.connect(art, 'norm_files', createfilter1, 'comp_norm')
wf.connect(art, 'outlier_files', createfilter1, 'outliers')

filter1 = MapNode(fsl.GLM(out_f_name='F_mcart.nii.gz',
                          out_pf_name='pF_mcart.nii.gz',
                          demean=True),
                  iterfield=['in_file', 'design', 'out_res_name'],
                  name='filtermotion')

wf.connect(realign, 'out_file', filter1, 'in_file')
wf.connect(realign, ('out_file', rename, '_filtermotart'),
           filter1, 'out_res_name')
wf.connect(createfilter1, 'out_files', filter1, 'design')

createfilter2 = MapNode(ACompCor(),
                       iterfield=['realigned_file', 'extra_regressors'],
                       name='makecompcorrfilter')
createfilter2.inputs.components_file = 'noise_components.txt'
createfilter2.inputs.num_components = num_components

wf.connect(createfilter1, 'out_files', createfilter2, 'extra_regressors')
wf.connect(filter1, 'out_res', createfilter2, 'realigned_file')
wf.connect(registration, ('outputspec.segmentation_files', selectindex, [0, 2]),
           createfilter2, 'mask_file')

filter2 = MapNode(fsl.GLM(out_f_name='F.nii.gz',
                          out_pf_name='pF.nii.gz',
                          demean=True),

```

```

        iterfield=['in_file', 'design', 'out_res_name'],
        name='filter_noise_nosmooth')
wf.connect(filter1, 'out_res', filter2, 'in_file')
wf.connect(filter1, ('out_res', rename, '_cleaned'),
            filter2, 'out_res_name')
wf.connect(createfilter2, 'components_file', filter2, 'design')
wf.connect(mask, 'mask_file', filter2, 'mask')

bandpass = Node(Function(input_names=['files', 'lowpass_freq',
                                     'highpass_freq', 'fs'],
                        output_names=['out_files'],
                        function=bandpass_filter,
                        imports=imports),
                name='bandpass_unsmooth')
bandpass.inputs.fs = 1. / TR
bandpass.inputs.highpass_freq = highpass_freq
bandpass.inputs.lowpass_freq = lowpass_freq
wf.connect(filter2, 'out_res', bandpass, 'files')

```

Smooth the functional data using `nipy.interfaces.fsl.IsotropicSmooth`.

```

smooth = MapNode(interface=fsl.IsotropicSmooth(), name="smooth", iterfield=["in_
↪file"])
smooth.inputs.fwhm = vol_fwhm

wf.connect(bandpass, 'out_files', smooth, 'in_file')

collector = Node(Merge(2), name='collect_streams')
wf.connect(smooth, 'out_file', collector, 'in1')
wf.connect(bandpass, 'out_files', collector, 'in2')

```

Transform the remaining images. First to anatomical and then to target

```

warpall = MapNode(ants.ApplyTransforms(), iterfield=['input_image'],
                  name='warpall')
warpall.inputs.input_image_type = 3
warpall.inputs.interpolation = 'Linear'
warpall.inputs.invert_transform_flags = [False, False]
warpall.inputs.terminal_output = 'file'
warpall.inputs.reference_image = target_file
warpall.inputs.args = '--float'
warpall.inputs.num_threads = 2
warpall.plugin_args = {'sbatch_args': '-c%d' % 2}

# transform to target
wf.connect(collector, 'out', warpall, 'input_image')
wf.connect(registration, 'outputspec.transforms', warpall, 'transforms')

mask_target = Node(fsl.ImageMaths(op_string='-bin'), name='target_mask')

wf.connect(registration, 'outputspec.anat2target', mask_target, 'in_file')

maskts = MapNode(fsl.ApplyMask(), iterfield=['in_file'], name='ts_masker')
wf.connect(warpall, 'output_image', maskts, 'in_file')
wf.connect(mask_target, 'out_file', maskts, 'mask_file')

# map to surface
# extract aparc+aseg ROIs
# extract subcortical ROIs

```

```

# extract target space ROIs
# combine subcortical and cortical rois into a single cifti file

#####
# Convert aparc to subject functional space

# Sample the average time series in aparc ROIs
sampleaparc = MapNode(freesurfer.SegStats(default_color_table=True),
                      iterfield=['in_file', 'summary_file',
                                'avgwf_txt_file'],
                      name='aparc_ts')
sampleaparc.inputs.segment_id = ([8] + list(range(10, 14)) + [17, 18, 26, 47] +
                                list(range(49, 55)) + [58] + list(range(1001,
↪1036)) +
                                list(range(2001, 2036)))

wf.connect(registration, 'outputspec.aparc',
           sampleaparc, 'segmentation_file')
wf.connect(collector, 'out', sampleaparc, 'in_file')

def get_names(files, suffix):
    """Generate appropriate names for output files
    """
    from nipy.utils.filemanip import (split_filename, filename_to_list,
                                     list_to_filename)

    import os
    out_names = []
    for filename in files:
        path, name, _ = split_filename(filename)
        out_names.append(os.path.join(path, name + suffix))
    return list_to_filename(out_names)

wf.connect(collector, ('out', get_names, '_avgwf.txt'),
           sampleaparc, 'avgwf_txt_file')
wf.connect(collector, ('out', get_names, '_summary.stats'),
           sampleaparc, 'summary_file')

# Sample the time series onto the surface of the target surface. Performs
# sampling into left and right hemisphere
target = Node(IdentityInterface(fields=['target_subject']), name='target')
target.iterables = ('target_subject', filename_to_list(target_subject))

samplerlh = MapNode(freesurfer.SampleToSurface(),
                    iterfield=['source_file'],
                    name='sampler_lh')
samplerlh.inputs.sampling_method = "average"
samplerlh.inputs.sampling_range = (0.1, 0.9, 0.1)
samplerlh.inputs.sampling_units = "frac"
samplerlh.inputs.interp_method = "trilinear"
samplerlh.inputs.smooth_surf = surf_fwhm
# samplerlh.inputs.cortex_mask = True
samplerlh.inputs.out_type = 'niigz'
samplerlh.inputs.subjects_dir = subjects_dir

samplerlh = samplerlh.clone('sampler_rh')

samplerlh.inputs.hemi = 'lh'
wf.connect(collector, 'out', samplerlh, 'source_file')

```

```

wf.connect(registration, 'outputspec.out_reg_file', samplerlh, 'reg_file')
wf.connect(target, 'target_subject', samplerlh, 'target_subject')

samplerlh.set_input('hemi', 'rh')
wf.connect(collector, 'out', samplerlh, 'source_file')
wf.connect(registration, 'outputspec.out_reg_file', samplerlh, 'reg_file')
wf.connect(target, 'target_subject', samplerlh, 'target_subject')

# Combine left and right hemisphere to text file
combiner = MapNode(Function(input_names=['left', 'right'],
                             output_names=['out_file'],
                             function=combine_hemi,
                             imports=imports),
                    iterfield=['left', 'right'],
                    name="combiner")
wf.connect(samplerlh, 'out_file', combiner, 'left')
wf.connect(samplerlh, 'out_file', combiner, 'right')

# Sample the time series file for each subcortical roi
ts2txt = MapNode(Function(input_names=['timeseries_file', 'label_file',
                                       'indices'],
                           output_names=['out_file'],
                           function=extract_subrois,
                           imports=imports),
                  iterfield=['timeseries_file'],
                  name='getsubcortts')
ts2txt.inputs.indices = [8] + list(range(10, 14)) + [17, 18, 26, 47] + \
    list(range(49, 55)) + [58]
ts2txt.inputs.label_file = \
    os.path.abspath(('OASIS-TRT-20_jointfusion_DKT31_CMA_labels_in_MNI152_'
                    '2mm_v2.nii.gz'))
wf.connect(maskts, 'out_file', ts2txt, 'timeseries_file')

#####

substitutions = [('_target_subject_', ''),
                 ('_filtermotart_cleaned_bp_trans_masked', ''),
                 ('_filtermotart_cleaned_bp', ''),
                 ]
substitutions += [("_smooth%d" % i, "") for i in range(11)[::-1]]
substitutions += [("_ts_masker%d" % i, "") for i in range(11)[::-1]]
substitutions += [("_getsubcortts%d" % i, "") for i in range(11)[::-1]]
substitutions += [("_combiner%d" % i, "") for i in range(11)[::-1]]
substitutions += [("_filtermotion%d" % i, "") for i in range(11)[::-1]]
substitutions += [("_filter_noise_nosmooth%d" % i, "") for i in range(11)[::-1]]
substitutions += [("_makecompcofilter%d" % i, "") for i in range(11)[::-1]]
substitutions += [("_get_aparc_tsnr%d/" % i, "run%d_" % (i + 1)) for i in
    range(11)[::-1]]

substitutions += [("T1_out_brain_pve_0_maths_warped", "compco_cs"),
                 ("T1_out_brain_pve_1_maths_warped", "compco_gm"),
                 ("T1_out_brain_pve_2_maths_warped", "compco_wm"),
                 ("output_warped_image_maths", "target_brain_mask"),
                 ("median_brain_mask", "native_brain_mask"),
                 ("corr_", "")]

regex_subs = [('_combiner.*/sar', '/smooth/'),
              ('_combiner.*/ar', '/unsmooth/'),

```

```

        ('_aparc_ts.*sar', '/smooth/'),
        ('_aparc_ts.*ar', '/unsmooth/'),
        ('_getsubcortts.*sar', '/smooth/'),
        ('_getsubcortts.*ar', '/unsmooth/'),
        ('series/sar', 'series/smooth/'),
        ('series/ar', 'series/unsmooth/'),
        ('_inverse_transform./', ''),
    ]
]
# Save the relevant data into an output directory
datasink = Node(interface=DataSink(), name="datasink")
datasink.inputs.base_directory = sink_directory
datasink.inputs.container = subject_id
datasink.inputs.substitutions = substitutions
datasink.inputs.regex_substitutions = regex_subs # (r'(/_.*(\d+))', r'/run\2')
wf.connect(realign, 'par_file', datasink, 'resting.qa.motion')
wf.connect(art, 'norm_files', datasink, 'resting.qa.art.@norm')
wf.connect(art, 'intensity_files', datasink, 'resting.qa.art.@intensity')
wf.connect(art, 'outlier_files', datasink, 'resting.qa.art.@outlier_files')
wf.connect(registration, 'outputspec.segmentation_files', datasink, 'resting.mask_
↪files')
wf.connect(registration, 'outputspec.anat2target', datasink, 'resting.qa.ants')
wf.connect(mask, 'mask_file', datasink, 'resting.mask_files.@brainmask')
wf.connect(mask_target, 'out_file', datasink, 'resting.mask_files.target')
wf.connect(filter1, 'out_f', datasink, 'resting.qa.compmaps.@mc_F')
wf.connect(filter1, 'out_pf', datasink, 'resting.qa.compmaps.@mc_pF')
wf.connect(filter2, 'out_f', datasink, 'resting.qa.compmaps')
wf.connect(filter2, 'out_pf', datasink, 'resting.qa.compmaps.@p')
wf.connect(registration, 'outputspec.min_cost_file', datasink, 'resting.qa.mincost
↪')
wf.connect(tsnr, 'tsnr_file', datasink, 'resting.qa.tsnr.@map')
wf.connect([(get_roi_tsnr, datasink, [('avgwf_txt_file', 'resting.qa.tsnr'),
↪('summary_file', 'resting.qa.tsnr.@summary
↪')])])

wf.connect(bandpass, 'out_files', datasink, 'resting.timeseries.@bandpassed')
wf.connect(smooth, 'out_file', datasink, 'resting.timeseries.@smoothed')
wf.connect(createfilter1, 'out_files',
        datasink, 'resting.regress.@regressors')
wf.connect(createfilter2, 'components_file',
        datasink, 'resting.regress.@compcorr')
wf.connect(maskts, 'out_file', datasink, 'resting.timeseries.target')
wf.connect(sampleaparc, 'summary_file',
        datasink, 'resting.parcellations.aparc')
wf.connect(sampleaparc, 'avgwf_txt_file',
        datasink, 'resting.parcellations.aparc.@avgwf')
wf.connect(ts2txt, 'out_file',
        datasink, 'resting.parcellations.grayo.@subcortical')

datasink2 = Node(interface=DataSink(), name="datasink2")
datasink2.inputs.base_directory = sink_directory
datasink2.inputs.container = subject_id
datasink2.inputs.substitutions = substitutions
datasink2.inputs.regex_substitutions = regex_subs # (r'(/_.*(\d+))', r'/run\2')
wf.connect(combiner, 'out_file',
        datasink2, 'resting.parcellations.grayo.@surface')
return wf

```

Creates the full workflow including getting information from dicom files

```

def create_resting_workflow(args, name=None):
    TR = args.TR
    slice_times = args.slice_times
    if args.dicom_file:
        TR, slice_times, slice_thickness = get_info(args.dicom_file)
        slice_times = (np.array(slice_times) / 1000.).tolist()

    if name is None:
        name = 'resting_' + args.subject_id
    kwargs = dict(files=[os.path.abspath(filename) for filename in args.files],
                  target_file=os.path.abspath(args.target_file),
                  subject_id=args.subject_id,
                  TR=TR,
                  slice_times=slice_times,
                  vol_fwhm=args.vol_fwhm,
                  surf_fwhm=args.surf_fwhm,
                  norm_threshold=2.,
                  subjects_dir=os.path.abspath(args.fsd_dir),
                  target_subject=args.target_surfs,
                  lowpass_freq=args.lowpass_freq,
                  highpass_freq=args.highpass_freq,
                  sink_directory=os.path.abspath(args.sink),
                  name=name)
    wf = create_workflow(**kwargs)
    return wf

if __name__ == "__main__":
    from argparse import ArgumentParser, RawTextHelpFormatter
    defstr = ' (default %(default)s)'
    parser = ArgumentParser(description=__doc__,
                            formatter_class=RawTextHelpFormatter)
    parser.add_argument("-d", "--dicom_file", dest="dicom_file",
                        help="a SIEMENS example dicom file from the resting series
→")
    parser.add_argument("-f", "--files", dest="files", nargs="+",
                        help="4d nifti files for resting state",
                        required=True)
    parser.add_argument("-t", "--target", dest="target_file",
                        help=("Target in MNI space. Best to use the MindBoggle "
                              "template - "
                              "OASIS-30_Atropos_template_in_MNI152_2mm.nii.gz"),
                        required=True)
    parser.add_argument("-s", "--subject_id", dest="subject_id",
                        help="FreeSurfer subject id", required=True)
    parser.add_argument("--subjects_dir", dest="fsdir",
                        help="FreeSurfer subject directory", required=True)
    parser.add_argument("--target_surfaces", dest="target_surfs", nargs="+",
                        default=['fsaverage5'],
                        help="FreeSurfer target surfaces" + defstr)
    parser.add_argument("--TR", dest="TR", default=None, type=float,
                        help="TR if dicom not provided in seconds")
    parser.add_argument("--slice_times", dest="slice_times", nargs="+",
                        type=float, help="Slice onset times in seconds")
    parser.add_argument("--vol_fwhm", default=6., dest='vol_fwhm',
                        type=float, help="Spatial FWHM" + defstr)
    parser.add_argument("--surf_fwhm", default=15., dest='surf_fwhm',
                        type=float, help="Spatial FWHM" + defstr)
    parser.add_argument("-l", "--lowpass_freq", dest="lowpass_freq",

```



```
        default=0.1, type=float,
        help="Low pass frequency (Hz)" + defstr)
parser.add_argument("-u", "--highpass_freq", dest="highpass_freq",
                    default=0.01, type=float,
                    help="High pass frequency (Hz)" + defstr)
parser.add_argument("-o", "--output_dir", dest="sink",
                    help="Output directory base", required=True)
parser.add_argument("-w", "--work_dir", dest="work_dir",
                    help="Output directory base")
parser.add_argument("-p", "--plugin", dest="plugin",
                    default='Linear',
                    help="Plugin to use")
parser.add_argument("--plugin_args", dest="plugin_args",
                    help="Plugin arguments")
args = parser.parse_args()

wf = create_resting_workflow(args)

if args.work_dir:
    work_dir = os.path.abspath(args.work_dir)
else:
    work_dir = os.getcwd()

wf.base_dir = work_dir
if args.plugin_args:
    wf.run(args.plugin, plugin_args=eval(args.plugin_args))
else:
    wf.run(args.plugin)
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.

Paper: Smoothing comparison

```

from builtins import range

import nipype.interfaces.io as nio           # Data i/o
import nipype.interfaces.spm as spm         # spm
import nipype.interfaces.freesurfer as fs    # freesurfer
import nipype.interfaces.nipy as nipy
import nipype.interfaces.utility as util
import nipype.pipeline.engine as pe         # pypeline engine
import nipype.algorithms.modelgen as model  # model specification
import nipype.workflows.fmri.fsl as fsl_wf
from nipype.interfaces.base import Bunch
import os                                   # system functions

preprocessing = pe.Workflow(name="preprocessing")

iter_fwhm = pe.Node(interface=util.IdentityInterface(fields=["fwhm"]),
                    name="iter_fwhm")
iter_fwhm.iterables = [('fwhm', [4, 8])]

iter_smoothing_method = pe.Node(interface=util.IdentityInterface(fields=[
    ↪ "smoothing_method"]),
                                name="iter_smoothing_method")
iter_smoothing_method.iterables = [('smoothing_method', ['isotropic_voxel',
                                                         'anisotropic_voxel',
                                                         'isotropic_surface'])]

realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True

isotropic_voxel_smooth = pe.Node(interface=spm.Smooth(),
                                name="isotropic_voxel_smooth")
preprocessing.connect(realign, "realigned_files", isotropic_voxel_smooth,
                    "in_files")
preprocessing.connect(iter_fwhm, "fwhm", isotropic_voxel_smooth, "fwhm")

compute_mask = pe.Node(interface=nipy.ComputeMask(), name="compute_mask")
preprocessing.connect(realign, "mean_image", compute_mask, "mean_volume")

```

```

anisotropic_voxel_smooth = fsl_wf.create_susan_smooth(name="anisotropic_voxel_
↪smooth",
                                                    separate_masks=False)
anisotropic_voxel_smooth.inputs.smooth.output_type = 'NIFTI'
preprocessing.connect(realign, "realigned_files", anisotropic_voxel_smooth,
                      "inputnode.in_files")
preprocessing.connect(iter_fwhm, "fwhm", anisotropic_voxel_smooth,
                      "inputnode.fwhm")
preprocessing.connect(compute_mask, "brain_mask", anisotropic_voxel_smooth,
                      'inputnode.mask_file')

recon_all = pe.Node(interface=fs.ReconAll(), name="recon_all")

surfregister = pe.Node(interface=fs.BBRegister(), name='surfregister')
surfregister.inputs.init = 'fsl'
surfregister.inputs.contrast_type = 't2'
preprocessing.connect(realign, 'mean_image', surfregister, 'source_file')
preprocessing.connect(recon_all, 'subject_id', surfregister, 'subject_id')
preprocessing.connect(recon_all, 'subjects_dir', surfregister, 'subjects_dir')

isotropic_surface_smooth = pe.MapNode(interface=fs.Smooth(proj_frac_avg=(0, 1, 0.
↪1)),
                                       iterfield=['in_file'],
                                       name="isotropic_surface_smooth")
preprocessing.connect(surfregister, 'out_reg_file', isotropic_surface_smooth,
                      'reg_file')
preprocessing.connect(realign, "realigned_files", isotropic_surface_smooth,
                      "in_file")
preprocessing.connect(iter_fwhm, "fwhm", isotropic_surface_smooth,
                      "surface_fwhm")
preprocessing.connect(iter_fwhm, "fwhm", isotropic_surface_smooth, "vol_fwhm")
preprocessing.connect(recon_all, 'subjects_dir', isotropic_surface_smooth,
                      'subjects_dir')

merge_smoothed_files = pe.Node(interface=util.Merge(3),
                                name='merge_smoothed_files')
preprocessing.connect(isotropic_voxel_smooth, 'smoothed_files',
                      merge_smoothed_files, 'in1')
preprocessing.connect(anisotropic_voxel_smooth, 'outputnode.smoothed_files',
                      merge_smoothed_files, 'in2')
preprocessing.connect(isotropic_surface_smooth, 'smoothed_file',
                      merge_smoothed_files, 'in3')

select_smoothed_files = pe.Node(interface=util.Select(),
                                name="select_smoothed_files")
preprocessing.connect(merge_smoothed_files, 'out', select_smoothed_files,
                      'inlist')

def chooseindex(roi):
    return {'isotropic_voxel': list(range(0, 4)), 'anisotropic_voxel': ↪
↪list(range(4, 8)),
           'isotropic_surface': list(range(8, 12))}[roi]

preprocessing.connect(iter_smoothing_method, ("smoothing_method", chooseindex),
                      select_smoothed_files, 'index')

```

```

rename = pe.MapNode(util.Rename(format_string="%(orig)s"), name="rename",
                    iterfield=['in_file'])
rename.inputs.parse_string = "(?P<orig>.*)"

preprocessing.connect(select_smoothed_files, 'out', rename, 'in_file')

specify_model = pe.Node(interface=model.SpecifyModel(), name="specify_model")
specify_model.inputs.input_units = 'secs'
specify_model.inputs.time_repetition = 3.
specify_model.inputs.high_pass_filter_cutoff = 120
specify_model.inputs.subject_info = [Bunch(conditions=['Task-Odd', 'Task-Even'],
                                           onsets=[list(range(15, 240, 60)),
                                                  list(range(45, 240, 60))],
                                           durations=[[15], [15]])] * 4

levelldesign = pe.Node(interface=spm.Level1Design(), name="levelldesign")
levelldesign.inputs.bases = {'hrf': {'derivs': [0, 0]}}
levelldesign.inputs.timing_units = 'secs'
levelldesign.inputs.interscan_interval = specify_model.inputs.time_repetition

levellestimate = pe.Node(interface=spm.EstimateModel(), name="levellestimate")
levellestimate.inputs.estimate_method = {'Classical': 1}

contrastestimate = pe.Node(interface=spm.EstimateContrast(),
                           name="contrastestimate")
contrastestimate.inputs.contrasts = [('Task>Baseline', 'T',
                                     ['Task-Odd', 'Task-Even'], [0.5, 0.5])]

modelling = pe.Workflow(name="modelling")
modelling.connect(specify_model, 'session_info', levelldesign, 'session_info')
modelling.connect(levelldesign, 'spm_mat_file', levellestimate, 'spm_mat_file')
modelling.connect(levellestimate, 'spm_mat_file', contrastestimate,
                  'spm_mat_file')
modelling.connect(levellestimate, 'beta_images', contrastestimate, 'beta_images')
modelling.connect(levellestimate, 'residual_image', contrastestimate,
                  'residual_image')

main_workflow = pe.Workflow(name="main_workflow")
main_workflow.base_dir = "smoothing_comparison_workflow"
main_workflow.connect(preprocessing, "realignment.realignment_parameters",
                     modelling, "specify_model.realignment_parameters")
main_workflow.connect(preprocessing, "select_smoothed_files.out",
                     modelling, "specify_model.functional_runs")
main_workflow.connect(preprocessing, "compute_mask.brain_mask",
                     modelling, "levelldesign.mask_image")

datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                              outfields=['func', 'struct']),
                    name='datasource')
datasource.inputs.base_directory = os.path.abspath('data')
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = info = dict(func=[['subject_id',
                                                    ['f3', 'f5', 'f7', 'f10']]],
                                              struct=[['subject_id', 'struct']])

datasource.inputs.subject_id = 's1'
datasource.inputs.sort_filelist = True

```

```
main_workflow.connect(datasource, 'func', preprocessing, 'realign.in_files')
main_workflow.connect(datasource, 'struct', preprocessing,
                      'recon_all.T1_files')

datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('smoothing_comparison_workflow/
↳output')
datasink.inputs.regexp_substitutions = [("_rename[0-9]", "")]

main_workflow.connect(modelling, 'contrastestimate.spmT_images', datasink,
                      'contrasts')
main_workflow.connect(preprocessing, 'rename.out_file', datasink,
                      'smoothed_epi')

main_workflow.run()
main_workflow.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

smRI: Using new ANTS for creating a T1 template

In this tutorial we will use ANTS (old version aka “ANTS”) based workflow to create a template out of multiple T1 volumes.

1. Tell python where to find the appropriate functions.

```
from __future__ import print_function, unicode_literals
from builtins import open
from future import standard_library
standard_library.install_aliases()

import os
import nipype.interfaces.utility as util
import nipype.interfaces.ants as ants
import nipype.interfaces.io as io
import nipype.pipeline.engine as pe  # pypeline engine

from nipype.workflows.smri.ants import ANTSTemplateBuildSingleIterationWF
```

2. Download T1 volumes into home directory

```
import urllib.request
import urllib.error
import urllib.parse
homeDir = os.getenv("HOME")
requestedPath = os.path.join(homeDir, 'nipypeTestPath')
mydatadir = os.path.realpath(requestedPath)
if not os.path.exists(mydatadir):
    os.makedirs(mydatadir)
print(mydatadir)

MyFileURLs = [
    ('http://slicer.kitware.com/midas3/download?bitstream=13121', '01_T1_half.nii.
↪gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13122', '02_T1_half.nii.
↪gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13124', '03_T1_half.nii.
↪gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13128', '01_T1_inv_half.
↪nii.gz'),
```

```

    ('http://slicer.kitware.com/midas3/download?bitstream=13123', '02_T1_inv_half.
↪nii.gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13125', '03_T1_inv_half.
↪nii.gz'),
]
for tt in MyFileURLs:
    myURL = tt[0]
    localFilename = os.path.join(mydatadir, tt[1])
    if not os.path.exists(localFilename):
        remotefile = urllib.request.urlopen(myURL)

        localFile = open(localFilename, 'wb')
        localFile.write(remotefile.read())
        localFile.close()
        print("Downloaded file: {0}".format(localFilename))
    else:
        print("File previously downloaded {0}".format(localFilename))

input_images = [
    os.path.join(mydatadir, '01_T1_half.nii.gz'),
    os.path.join(mydatadir, '02_T1_half.nii.gz'),
    os.path.join(mydatadir, '03_T1_half.nii.gz')
]
input_passive_images = [
    {'INV_T1': os.path.join(mydatadir, '01_T1_inv_half.nii.gz')},
    {'INV_T1': os.path.join(mydatadir, '02_T1_inv_half.nii.gz')},
    {'INV_T1': os.path.join(mydatadir, '03_T1_inv_half.nii.gz')}
]

```

3. Define the workflow and its working directory

```

tbuilder = pe.Workflow(name="ANTSTemplateBuilder")
tbuilder.base_dir = requestedPath

```

4. Define data sources. In real life these would be replace by DataGrabbers

```

datasource = pe.Node(interface=util.IdentityInterface(fields=['imageList',
↪'passiveImagesDictionariesList']),
                      run_without_submitting=True,
                      name='InputImages')
datasource.inputs.imageList = input_images
datasource.inputs.passiveImagesDictionariesList = input_passive_images
datasource.inputs.sort_filelist = True

```

5. Template is initialized by a simple average

```

initAvg = pe.Node(interface=ants.AverageImages(), name='initAvg')
initAvg.inputs.dimension = 3
initAvg.inputs.normalize = True

tbuilder.connect(datasource, "imageList", initAvg, "images")

```

6. Define the first iteration of template building

```

buildTemplateIteration1 = ANTSTemplateBuildSingleIterationWF('iteration01')
tbuilder.connect(initAvg, 'output_average_image', buildTemplateIteration1,
↪'inputspec.fixed_image')
tbuilder.connect(datasource, 'imageList', buildTemplateIteration1, 'inputspec.
↪images')
tbuilder.connect(datasource, 'passiveImagesDictionariesList',
↪buildTemplateIteration1, 'inputspec.ListOfPassiveImagesDictionaries')

```

7. Define the second iteration of template building

```
buildTemplateIteration2 = ANTSTemplateBuildSingleIterationWF('iteration02')
tbuilder.connect(buildTemplateIteration1, 'outputspec.template',
↳ buildTemplateIteration2, 'inputspec.fixed_image')
tbuilder.connect(datasource, 'imageList', buildTemplateIteration2, 'inputspec.
↳ images')
tbuilder.connect(datasource, 'passiveImagesDictionariesList',
↳ buildTemplateIteration2, 'inputspec.ListOfPassiveImagesDictionaries')
```

8. Move selected files to a designated results folder

```
datasink = pe.Node(io.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.join(requestedPath, "results")

tbuilder.connect(buildTemplateIteration2, 'outputspec.template', datasink,
↳ 'PrimaryTemplate')
tbuilder.connect(buildTemplateIteration2, 'outputspec.passive_deformed_templates',
↳ datasink, 'PassiveTemplate')
tbuilder.connect(initAvg, 'output_average_image', datasink, 'PreRegisterAverage')
```

8. Run the workflow

```
tbuilder.run()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

sMRI: Using ANTS for registration

In this simple tutorial we will use the Registration interface from ANTS to coregister two T1 volumes.

1. Tell python where to find the appropriate functions.

```
from __future__ import print_function, unicode_literals
from builtins import open

from future import standard_library
standard_library.install_aliases()

import os
import urllib.request
import urllib.error
import urllib.parse
from nipype.interfaces.ants import Registration
from nipype.testing import example_data
```

2. Download T1 volumes into home directory

```
homeDir = os.getenv("HOME")
requestedPath = os.path.join(homeDir, 'nipypeTestPath')
mydatadir = os.path.realpath(requestedPath)
if not os.path.exists(mydatadir):
    os.makedirs(mydatadir)
print(mydatadir)

MyFileURLs = [
    ('http://slicer.kitware.com/midas3/download?bitstream=13121', '01_T1_half.nii.
↪gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13122', '02_T1_half.nii.
↪gz'),
]
for tt in MyFileURLs:
    myURL = tt[0]
    localFilename = os.path.join(mydatadir, tt[1])
    if not os.path.exists(localFilename):
        remotefile = urllib.request.urlopen(myURL)

        localFile = open(localFilename, 'wb')
```

```
        localFile.write(remotefile.read())
        localFile.close()
        print("Downloaded file: {0}".format(localFilename))
    else:
        print("File previously downloaded {0}".format(localFilename))

input_images = [
    os.path.join(mydatadir, '01_T1_half.nii.gz'),
    os.path.join(mydatadir, '02_T1_half.nii.gz'),
]
```

3. Define the parameters of the registration. Settings are found in the file `smri_ants_registration_settings.json` distributed with the `example_data` of *nipyype*.

```
reg = Registration(from_file=example_data('smri_ants_registration_settings.json'))
reg.inputs.fixed_image = input_images[0]
reg.inputs.moving_image = input_images[1]
```

Alternatively to the use of the `from_file` feature to load ANTs settings, the user can manually set all those inputs instead:

```
reg.inputs.output_transform_prefix = 'thisTransform'
reg.inputs.output_warped_image = 'INTERNAL_WARPED.nii.gz'
reg.inputs.output_transform_prefix = "output_"
reg.inputs.transforms = ['Translation', 'Rigid', 'Affine', 'SyN']
reg.inputs.transform_parameters = [(0.1,), (0.1,), (0.1,), (0.2, 3.0, 0.0)]
reg.inputs.number_of_iterations = ([[10000, 11110, 11110]] * 3 +
                                   [[100, 50, 30]])

reg.inputs.dimension = 3
reg.inputs.write_composite_transform = True
reg.inputs.collapse_output_transforms = False
reg.inputs.metric = ['Mattes'] * 3 + [['Mattes', 'CC']]
reg.inputs.metric_weight = [1] * 3 + [[0.5, 0.5]]
reg.inputs.radius_or_number_of_bins = [32] * 3 + [[32, 4]]
reg.inputs.sampling_strategy = ['Regular'] * 3 + [[None, None]]
reg.inputs.sampling_percentage = [0.3] * 3 + [[None, None]]
reg.inputs.convergence_threshold = [1.e-8] * 3 + [-0.01]
reg.inputs.convergence_window_size = [20] * 3 + [5]
reg.inputs.smoothing_sigmas = [[4, 2, 1]] * 3 + [[1, 0.5, 0]]
reg.inputs.sigma_units = ['vox'] * 4
reg.inputs.shrink_factors = [[6, 4, 2]] + [[3, 2, 1]] * 2 + [[4, 2, 1]]
reg.inputs.use_estimate_learning_rate_once = [True] * 4
reg.inputs.use_histogram_matching = [False] * 3 + [True]
reg.inputs.initial_moving_transform_com = True
```

```
print(reg.cmdline)
```

3. Run the registration

```
reg.run()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipyype source distribution under the `examples` directory.

smRI: Using new ANTS for creating a T1 template (ITK4)

In this tutorial we will use ANTS (new ITK4 version aka “antsRegistration”) based workflow to create a template out of multiple T1 volumes. We will also showcase how to fine tune SGE jobs requirements.

1. Tell python where to find the appropriate functions.

```
from __future__ import print_function
from future import standard_library
standard_library.install_aliases()

import os
import nipype.interfaces.utility as util
import nipype.interfaces.ants as ants
import nipype.interfaces.io as io
import nipype.pipeline.engine as pe  # pypeline engine

from nipype.workflows.smri.ants import _
↪antsRegistrationTemplateBuildSingleIterationWF
```

2. Download T1 volumes into home directory

```
import urllib.request
import urllib.error
import urllib.parse
homeDir = os.getenv("HOME")
requestedPath = os.path.join(homeDir, 'nipypeTestPath')
mydatadir = os.path.realpath(requestedPath)
if not os.path.exists(mydatadir):
    os.makedirs(mydatadir)
print(mydatadir)

MyFileURLs = [
    ('http://slicer.kitware.com/midas3/download?bitstream=13121', '01_T1_half.nii.
↪gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13122', '02_T1_half.nii.
↪gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13124', '03_T1_half.nii.
↪gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13128', '01_T1_inv_half.
↪nii.gz'),
```

```
    ('http://slicer.kitware.com/midas3/download?bitstream=13123', '02_T1_inv_half.
↪nii.gz'),
    ('http://slicer.kitware.com/midas3/download?bitstream=13125', '03_T1_inv_half.
↪nii.gz'),
]
for tt in MyFileURLs:
    myURL = tt[0]
    localFilename = os.path.join(mydatadir, tt[1])
    if not os.path.exists(localFilename):
        remotefile = urllib.request.urlopen(myURL)

        localFile = open(localFilename, 'wb')
        localFile.write(remotefile.read())
        localFile.close()
        print("Downloaded file: {}".format(localFilename))
    else:
        print("File previously downloaded {}".format(localFilename))
```

ListOfImagesDictionaries - a list of dictionaries where each dictionary is for one scan session, and the mappings in the dictionary are for all the co-aligned images for that one scan session

```
ListOfImagesDictionaries = [
    {'T1': os.path.join(mydatadir, '01_T1_half.nii.gz'), 'INV_T1': os.path.
↪join(mydatadir, '01_T1_inv_half.nii.gz'), 'LABEL_MAP': os.path.join(mydatadir,
↪'01_T1_inv_half.nii.gz')},
    {'T1': os.path.join(mydatadir, '02_T1_half.nii.gz'), 'INV_T1': os.path.
↪join(mydatadir, '02_T1_inv_half.nii.gz'), 'LABEL_MAP': os.path.join(mydatadir,
↪'02_T1_inv_half.nii.gz')},
    {'T1': os.path.join(mydatadir, '03_T1_half.nii.gz'), 'INV_T1': os.path.
↪join(mydatadir, '03_T1_inv_half.nii.gz'), 'LABEL_MAP': os.path.join(mydatadir,
↪'03_T1_inv_half.nii.gz')}
]
input_passive_images = [
    {'INV_T1': os.path.join(mydatadir, '01_T1_inv_half.nii.gz')},
    {'INV_T1': os.path.join(mydatadir, '02_T1_inv_half.nii.gz')},
    {'INV_T1': os.path.join(mydatadir, '03_T1_inv_half.nii.gz')}
]
```

registrationImageTypes - A list of the image types to be used actively during the estimation process of registration, any image type not in this list will be passively resampled with the estimated transforms. ['T1', 'T2']

```
registrationImageTypes = ['T1']
```

interpolationMap - A map of image types to interpolation modes. If an image type is not listed, it will be linearly interpolated. { 'labelmap': 'NearestNeighbor', 'FLAIR': 'WindowedSinc' }

```
interpolationMapping = {'INV_T1': 'LanczosWindowedSinc', 'LABEL_MAP':
↪'NearestNeighbor', 'T1': 'Linear'}
```

3. Define the workflow and its working directory

```
tbuilder = pe.Workflow(name="antsRegistrationTemplateBuilder")
tbuilder.base_dir = requestedPath
```

4. Define data sources. In real life these would be replace by DataGrabbers

```
InitialTemplateInputs = [mdict['T1'] for mdict in ListOfImagesDictionaries]

datasource = pe.Node(interface=util.IdentityInterface(fields=[
↪'InitialTemplateInputs', 'ListOfImagesDictionaries',
```

```

↪'registrationImageTypes', 'interpolationMapping']],
    run_without_subsubmitting=True,
    name='InputImages')
datasource.inputs.InitialTemplateInputs = InitialTemplateInputs
datasource.inputs.ListOfImagesDictionaries = ListOfImagesDictionaries
datasource.inputs.registrationImageTypes = registrationImageTypes
datasource.inputs.interpolationMapping = interpolationMapping
datasource.inputs.sort_filelist = True

```

5. Template is initialized by a simple average in this simple example, any reference image could be used (i.e. a previously created template)

```

initAvg = pe.Node(interface=ants.AverageImages(), name='initAvg')
initAvg.inputs.dimension = 3
initAvg.inputs.normalize = True

tbuilder.connect(datasource, "InitialTemplateInputs", initAvg, "images")

```

6. Define the first iteration of template building

```

buildTemplateIteration1 = antsRegistrationTemplateBuildSingleIterationWF(
↪'iteration01')

```

Here we are fine tuning parameters of the SGE job (memory limit, numebr of cores etc.)

```

BeginANTS = buildTemplateIteration1.get_node("BeginANTS")
BeginANTS.plugin_args = {'qsub_args': '-S /bin/bash -pe smp1 8-12 -l mem_
↪free=6000M -o /dev/null -e /dev/null queue_name', 'overwrite': True}

tbuilder.connect(initAvg, 'output_average_image', buildTemplateIteration1,
↪'inputspec.fixed_image')
tbuilder.connect(datasource, 'ListOfImagesDictionaries', buildTemplateIteration1,
↪'inputspec.ListOfImagesDictionaries')
tbuilder.connect(datasource, 'registrationImageTypes', buildTemplateIteration1,
↪'inputspec.registrationImageTypes')
tbuilder.connect(datasource, 'interpolationMapping', buildTemplateIteration1,
↪'inputspec.interpolationMapping')

```

7. Define the second iteration of template building

```

buildTemplateIteration2 = antsRegistrationTemplateBuildSingleIterationWF(
↪'iteration02')
BeginANTS = buildTemplateIteration2.get_node("BeginANTS")
BeginANTS.plugin_args = {'qsub_args': '-S /bin/bash -pe smp1 8-12 -l mem_
↪free=6000M -o /dev/null -e /dev/null queue_name', 'overwrite': True}
tbuilder.connect(buildTemplateIteration1, 'outputspec.template',
↪buildTemplateIteration2, 'inputspec.fixed_image')
tbuilder.connect(datasource, 'ListOfImagesDictionaries', buildTemplateIteration2,
↪'inputspec.ListOfImagesDictionaries')
tbuilder.connect(datasource, 'registrationImageTypes', buildTemplateIteration2,
↪'inputspec.registrationImageTypes')
tbuilder.connect(datasource, 'interpolationMapping', buildTemplateIteration2,
↪'inputspec.interpolationMapping')

```

8. Move selected files to a designated results folder

```

datasink = pe.Node(io.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.join(requestedPath, "results")

tbuilder.connect(buildTemplateIteration2, 'outputspec.template', datasink,
↪'PrimaryTemplate')

```

```
tbuilder.connect(buildTemplateIteration2, 'outputspec.passive_deformed_templates',  
↳ datasink, 'PassiveTemplate')  
tbuilder.connect(initAvg, 'output_average_image', datasink, 'PreRegisterAverage')
```

9. Run the workflow

```
tbuilder.run(plugin="SGE")
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

sMRI: USing CBS Tools for skullstripping

This simple workflow uses SPECTRE2010 algorithm to skullstrip an MP2RAGE anatomical scan.

```
import nipype.pipeline.engine as pe
from nipype.interfaces.mipav.developer import JistIntensityMp2rageMasking, \
↳ MedicAlgorithmSPECTRE2010

wf = pe.Workflow("skullstripping")

mask = pe.Node(JistIntensityMp2rageMasking(), name="masking")
mask.inputs.inSecond = "/Users/filo/7t_trt/niftis/sub001/session_1/MP2RAGE_INV2.
↳ nii.gz"
mask.inputs.inQuantitative = "/Users/filo/7t_trt/niftis/sub001/session_1/MP2RAGE_
↳ UNI.nii.gz"
mask.inputs.inT1weighted = "/Users/filo/7t_trt/niftis/sub001/session_1/MP2RAGE_T1.
↳ nii.gz"
mask.inputs.outMasked = True
mask.inputs.outMasked2 = True
mask.inputs.outSignal = True
mask.inputs.outSignal2 = True

skullstrip = pe.Node(MedicAlgorithmSPECTRE2010(), name="skullstrip")
skullstrip.inputs.outStripped = True
skullstrip.inputs.xDefaultMem = 6000

wf.connect(mask, 'outMasked', skullstrip, 'inInput')
wf.run()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.

sMRI: FreeSurfer

This script, `smri_freesurfer.py`, demonstrates the ability to call `reconall` on a set of subjects and then make an average subject:

```
python smri_freesurfer.py
```

Import necessary modules from `nipype`.

```
import os

import nipype.pipeline.engine as pe
import nipype.interfaces.io as nio
from nipype.interfaces.freesurfer.preprocess import ReconAll
from nipype.interfaces.freesurfer.utils import MakeAverageSubject

subject_list = ['s1', 's3']
data_dir = os.path.abspath('data')
subjects_dir = os.path.abspath('amri_freesurfer_tutorial/subjects_dir')

wf = pe.Workflow(name="11workflow")
wf.base_dir = os.path.abspath('amri_freesurfer_tutorial/workdir')
```

Grab data

```
datasource = pe.MapNode(interface=nio.DataGrabber(infields=['subject_id'],
                                                    outfields=['struct']),
                        name='datasource',
                        iterfield=['subject_id'])
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = dict(struct=[['subject_id', 'struct']])
datasource.inputs.subject_id = subject_list
datasource.inputs.sort_filelist = True
```

Run recon-all

```
recon_all = pe.MapNode(interface=ReconAll(), name='recon_all',
                      iterfield=['subject_id', 'T1_files'])
recon_all.inputs.subject_id = subject_list
```

```
if not os.path.exists(subjects_dir):
    os.mkdir(subjects_dir)
recon_all.inputs.subjects_dir = subjects_dir

wf.connect(datasource, 'struct', recon_all, 'T1_files')
```

Make average subject

```
average = pe.Node(interface=MakeAverageSubject(), name="average")
average.inputs.subjects_dir = subjects_dir

wf.connect(recon_all, 'subject_id', average, 'subjects_ids')

wf.run("MultiProc", plugin_args={'n_procs': 4})
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

sMRI: FSReconAll

This script, `smri_fsreconall.py`, demonstrates the ability to use the `create_reconall_workflow` function to create a workflow and then run it on a set of subjects and then make an average subject:

```
python smri_fsreconall.py
```

For an example on how to call FreeSurfer's reconall script in Nipype see `smri_freesurfer.py`.
Import necessary modules from nipype.

```
import os

import nipype.pipeline.engine as pe
import nipype.interfaces.io as nio
from nipype.workflows.smri.freesurfer import create_reconall_workflow
from nipype.interfaces.freesurfer.utils import MakeAverageSubject
from nipype.interfaces.utility import IdentityInterface
```

Assign the tutorial directory

```
tutorial_dir = os.path.abspath('smri_fsreconall_tutorial')
if not os.path.isdir(tutorial_dir):
    os.mkdir(tutorial_dir)
```

Define the workflow directories

```
subject_list = ['s1', 's3']
data_dir = os.path.abspath('data')
subjects_dir = os.path.join(tutorial_dir, 'subjects_dir')
if not os.path.exists(subjects_dir):
    os.mkdir(subjects_dir)

wf = pe.Workflow(name="11workflow")
wf.base_dir = os.path.join(tutorial_dir, 'workdir')
```

Create inputspec

```
inputspec = pe.Node(interface=IdentityInterface(['subject_id']),
                    name="inputspec")
inputspec.iterables = ("subject_id", subject_list)
```

Grab data

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                                outfields=['struct']),
                      name='datasource')
datasource.inputs.base_directory = data_dir
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = dict(struct=[['subject_id', 'struct']])
datasource.inputs.subject_id = subject_list
datasource.inputs.sort_filelist = True

wf.connect(inputspec, 'subject_id', datasource, 'subject_id')
```

Run recon-all

```
recon_all = create_reconall_workflow()
recon_all.inputs.inputspec.subjects_dir = subjects_dir

wf.connect(datasource, 'struct', recon_all, 'inputspec.T1_files')
wf.connect(inputspec, 'subject_id', recon_all, 'inputspec.subject_id')
```

Make average subject

```
average = pe.JoinNode(interface=MakeAverageSubject(),
                      joinsource="inputspec",
                      joinfield="subjects_ids",
                      name="average")
average.inputs.subjects_dir = subjects_dir

wf.connect(recon_all, 'postdatasink_outputspec.subject_id', average, 'subjects_ids
→')

wf.run("MultiProc", plugin_args={'n_procs': 4})
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the examples directory.

sMRI: Regional Tessellation and Surface Smoothing

Introduction

This script, `tessellation_tutorial.py`, demonstrates the use of `create_tessellation_flow` from `nipype.workflows.smri.freesurfer`, and it can be run with:

```
python tessellation_tutorial.py
```

This example requires that the user has Freesurfer installed, and that the Freesurfer directory for 'fsaverage' is present.

See also:

ConnectomeViewer The Connectome Viewer connects Multi-Modal Multi-Scale Neuroimaging and Network Datasets For Analysis and Visualization in Python.

<http://www.geuz.org/gmsh/> Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities

<http://www.blender.org/> Blender is the free open source 3D content creation suite, available for all major operating systems under the GNU General Public License.

Warning: This workflow will take several hours to finish entirely, since smoothing the larger cortical surfaces is very time consuming.

Packages and Data Setup

Import the necessary modules and workflow from `nipype`.

```
import nipype.pipeline.engine as pe          # pypeline engine
import nipype.interfaces.cmtk as cmtk
import nipype.interfaces.io as nio          # Data i/o
import os
import os.path as op
from nipype.workflows.smri.freesurfer import create_tessellation_flow
```

Directories

Set the default directory and lookup table (LUT) paths

```
fs_dir = os.environ['FREESURFER_HOME']
lookup_file = op.join(fs_dir, 'FreeSurferColorLUT.txt')
subjects_dir = op.join(fs_dir, 'subjects/')
output_dir = './tessellate_tutorial'
```

Inputs

Create the tessellation workflow and set inputs Here we will choose Gifti (gii) as the output format, because we want to be able to view the surface in ConnectomeViewer.

In you intend to view the meshes in gmsh or Blender, you should change the workflow creation to use stereolithographic (stl) format.

```
tessflow = create_tessellation_flow(name='tessflow', out_format='gii')
tessflow.inputs.inputs.spec.subject_id = 'fsaverage'
tessflow.inputs.inputs.spec.subjects_dir = subjects_dir
tessflow.inputs.inputs.spec.lookup_file = lookup_file
```

We also create a conditional node to package the surfaces for ConnectomeViewer. Simply set cff to “False” to ignore this step.

```
cff = True
if cff:
    cff = pe.Node(interface=cmtk.CFFConverter(), name='cff')
    cff.inputs.out_file = 'Meshes.cff'
```

Outputs

Create a datasink to organize the smoothed meshes Using regular-expression substitutions we can remove the extraneous folders generated by the mapnode.

```
datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = 'meshes'
datasink.inputs.regexp_substitutions = [('_smoother[\d]*\/', '')]
```

Execution

Finally, create and run another pipeline that connects the workflow and datasink

```
tesspipe = pe.Workflow(name='tessellate_tutorial')
tesspipe.base_dir = output_dir
tesspipe.connect([(tessflow, datasink, [(['outputs.spec.meshes', '@meshes.all'])])])
```

If the surfaces are to be packaged, this will connect the CFFConverter node to the tessellation and smoothing workflow, as well as to the datasink.

```
if cff:
    tesspipe.connect([(tessflow, cff, [(['outputs.spec.meshes', 'gifti_surfaces'])])])
    tesspipe.connect([(cff, datasink, [(['connectome_file', '@cff'])])])

tesspipe.run()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipyte source distribution under the `examples` directory.

Example source code

You can download the full source code of this example. This same script is also included in the Nipyte source distribution under the `examples` directory.

Workflow from scratch

```

from builtins import range

import nipype.interfaces.io as nio           # Data i/o
import nipype.interfaces.spm as spm          # spm
import nipype.pipeline.engine as pe          # pypeline engine
import nipype.algorithms.modelgen as model   # model specification
from nipype.interfaces.base import Bunch
import os                                     # system functions

```

In the following section, to showcase NiPyPe, we will describe how to create and extend a typical fMRI processing pipeline. We will begin with a basic processing layout and follow with extending it by adding/exchanging different components.

Most fMRI pipeline can be divided into two sections - preprocessing and modelling. First one deals with cleaning data from confounds and noise and the second one fits a model based on the experimental design. Preprocessing stage in our first iteration of a pipeline will consist of only two steps: realignment and smoothing. In NiPyPe Every processing step consist of an Interface (which defines how to execute corresponding software) encapsulated in a Node (which defines for example a unique name). For realignment (motion correction achieved by coregistering all volumes to the mean) and smoothing (convolution with 3D Gaussian kernel) we will use SPM implementation. Definition of appropriate nodes can be found in Listing 1 (TODO). Inputs (such as `register_to_mean` from listing 1) of nodes are accessible through the `inputs` property. Upon setting any input its type is verified to avoid errors during the execution.

```

realign = pe.Node(interface=spm.Realign(), name="realign")
realign.inputs.register_to_mean = True

smooth = pe.Node(interface=spm.Smooth(), name="smooth")
smooth.inputs.fwhm = 4

```

To connect two nodes a Workflow has to be created. `connect()` method of a Workflow allows to specify which outputs of which Nodes should be connected to which inputs of which Nodes (see Listing 2). By connecting `realigned_files` output of `realign` to `in_files` input of `Smooth` we have created a simple preprocessing workflow (see Figure TODO).

```

preprocessing = pe.Workflow(name="preprocessing")
preprocessing.connect(realign, "realigned_files", smooth, "in_files")

```

Creating a modelling workflow which will define the design, estimate model and contrasts follows the same

suite. We will again use SPM implementations. NiPyPe, however, adds extra abstraction layer to model definition which allows using the same definition for many model estimation implementations (for example one from FSL or nippy). Therefore we will need four nodes: SpecifyModel (NiPyPe specific abstraction layer), Level1Design (SPM design definition), ModelEstimate, and ContrastEstimate. The connected modelling Workflow can be seen on Figure TODO. Model specification supports block, event and sparse designs. Contrasts provided to ContrastEstimate are defined using the same names of regressors as defined in the SpecifyModel.

```
specify_model = pe.Node(interface=model.SpecifyModel(), name="specify_model")
specify_model.inputs.input_units = 'secs'
specify_model.inputs.time_repetition = 3.
specify_model.inputs.high_pass_filter_cutoff = 120
specify_model.inputs.subject_info = [Bunch(conditions=['Task-Odd', 'Task-Even'],
                                             onsets=[list(range(15, 240, 60)),
                                                    list(range(45, 240, 60))],
                                             durations=[[15], [15]])] * 4

level1design = pe.Node(interface=spm.Level1Design(), name="level1design")
level1design.inputs.bases = {'hrf': {'derivs': [0, 0]}}
level1design.inputs.timing_units = 'secs'
level1design.inputs.interscan_interval = specify_model.inputs.time_repetition

level1estimate = pe.Node(interface=spm.EstimateModel(), name="level1estimate")
level1estimate.inputs.estimate_method = {'Classical': 1}

contrastestimate = pe.Node(interface=spm.EstimateContrast(),
                           name="contrastestimate")
cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
contrastestimate.inputs.contrasts = [cont1, cont2]

modelling = pe.Workflow(name="modelling")
modelling.connect(specify_model, 'session_info', level1design, 'session_info')
modelling.connect(level1design, 'spm_mat_file', level1estimate, 'spm_mat_file')
modelling.connect(level1estimate, 'spm_mat_file',
                  contrastestimate, 'spm_mat_file')
modelling.connect(level1estimate, 'beta_images', contrastestimate, 'beta_images')
modelling.connect(level1estimate, 'residual_image',
                  contrastestimate, 'residual_image')
```

Having preprocessing and modelling workflows we need to connect them together, add data grabbing facility and save the results. For this we will create a master Workflow which will host preprocessing and model Workflows as well as DataGrabber and DataSink Nodes. NiPyPe allows connecting Nodes between Workflows. We will use this feature to connect realignment_parameters and smoothed_files to modelling workflow.

```
main_workflow = pe.Workflow(name="main_workflow")
main_workflow.base_dir = "workflow_from_scratch"
main_workflow.connect(preprocessing, "realignment_parameters",
                     modelling, "specify_model.realignment_parameters")
main_workflow.connect(preprocessing, "smooth.smoothed_files",
                     modelling, "specify_model.functional_runs")
```

DataGrabber allows to define flexible search patterns which can be parameterized by user defined inputs (such as subject ID, session etc.). This allows to adapt to a wide range of file layouts. In our case we will parameterize it with subject ID. In this way we will be able to run it for different subjects. We can automate this by iterating over a list of subject IDs, by setting an iterables property on the subject_id input of DataGrabber. Its output will be connected to realignment node from preprocessing workflow.

```
datasource = pe.Node(interface=nio.DataGrabber(infields=['subject_id'],
                                             outfields=['func']),
```

```
        name='datasource')
datasource.inputs.base_directory = os.path.abspath('data')
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.template_args = dict(func=[['subject_id',
                                              ['f3', 'f5', 'f7', 'f10']]])
datasource.inputs.subject_id = 's1'
datasource.inputs.sort_filelist = True

main_workflow.connect(datasource, 'func', preprocessing, 'realigned_files')
```

DataSink on the other side provides means to storing selected results to a specified location. It supports automatic creation of folder stricter and regular expression based substitutions. In this example we will store T maps.

```
datasink = pe.Node(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('workflow_from_scratch/output')

main_workflow.connect(modelling, 'contrastestimate.spmT_images',
                      datasink, 'contrasts.@T')

main_workflow.run()
main_workflow.write_graph()
```

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

Workshop: Dartmouth College 2010

First lets go to the directory with the data we'll be working on and start the interactive python interpreter (with some nipype specific configuration). Note that nipype does not need to be run through ipython - it is just much nicer to do interactive work in it.

```
cd $TDPATH
ipython -p nipype
```

For every neuroimaging procedure supported by nipype there exists a wrapper - a small piece of code managing the underlying software (FSL, SPM, AFNI etc.). We call those interfaces. They are standardised so we can hook them up together. Lets have a look at some of them.

```
In [1]: import nipype.interfaces.fsl as fsl

In [2]: fsl.BET.help()
Inputs
-----

Mandatory:
  in_file: input file to skull strip

Optional:
  args: Additional parameters to the command
  center: center of gravity in voxels
  environ: Environment variables (default={})
  frac: fractional intensity threshold
  functional: apply to 4D fMRI data
  mutually_exclusive: functional, reduce_bias
  mask: create binary mask image
  mesh: generate a vtk mesh brain surface
  no_output: Don't generate segmented output
  out_file: name of output skull stripped image
  outline: create surface outline image
  output_type: FSL output type
  radius: head radius
  reduce_bias: bias field and neck cleanup
  mutually_exclusive: functional, reduce_bias
  skull: create skull image
  threshold: apply thresholding to segmented brain image and mask
```

```
vertical_gradient: vertical gradient in fractional intensity threshold (-1, 1)

Outputs
-----
mask_file: path/name of binary brain mask (if generated)
meshfile: path/name of vtk mesh file (if generated)
out_file: path/name of skullstripped file
outline_file: path/name of outline file (if generated)

In [3]: import nipy.interfaces.freesurfer as fs

In [4]: fs.Smooth.help()
Inputs
-----

Mandatory:
in_file: source volume
num_iters: number of iterations instead of fwhm
    mutually exclusive: surface_fwhm
reg_file: registers volume to surface anatomical
surface_fwhm: surface FWHM in mm
    mutually exclusive: num_iters
requires: reg_file

Optional:
args: Additional parameters to the command
environ: Environment variables (default={})
proj_frac: project frac of thickness a long surface normal
    mutually exclusive: proj_frac_avg
proj_frac_avg: average a long normal min max delta
    mutually exclusive: proj_frac
smoothed_file: output volume
subjects_dir: subjects directory
vol_fwhm: volumesmoothing outside of surface

Outputs
-----
args: Additional parameters to the command
environ: Environment variables
smoothed_file: smoothed input volume
subjects_dir: subjects directory
```

You can read about all of the interfaces implemented in nipy at our online documentation at <http://nipy.sourceforge.net/nipy/documentation.html#documentation> . Check it out now.

Using interfaces

Having interfaces allows us to use third party software (like FSL BET) as function. Look how simple it is.

```
from __future__ import print_function
from builtins import str

import nipy.interfaces.fsl as fsl
result = fsl.BET(in_file='data/s1/struct.nii').run()
print(result)
```

Running a single program is not much of a breakthrough. Lets run motion correction followed by smoothing (isotropic - in other words not using SUSAN). Notice that in the first line we are setting the output data type for

all FSL interfaces.

```
fsl.FSLCommand.set_default_output_type('NIFTI_GZ')
result1 = fsl.MCFLIRT(in_file='data/s1/f3.nii').run()
result2 = fsl.Smooth(in_file='f3_mcf.nii.gz', fwhm=6).run()
```

Simple workflow

In the previous example we knew that `fsl.MCFLIRT` will produce a file called `f3_mcf.nii.gz` and we have hard coded this as an input to `fsl.Smooth`. This is quite limited, but luckily `nipype` supports joining interfaces in pipelines. This way output of one interface will be used as an input of another without having to hard code anything. Before connecting Interfaces we need to put them into (separate) Nodes and give them unique names. This way every interface will process data in a separate folder.

```
import nipype.pipeline.engine as pe
import os

motion_correct = pe.Node(interface=fsl.MCFLIRT(in_file=os.path.abspath('data/s1/
↳f3.nii')),
                        name="motion_correct")
smooth = pe.Node(interface=fsl.Smooth(fwhm=6), name="smooth")

motion_correct_and_smooth = pe.Workflow(name="motion_correct_and_smooth")
motion_correct_and_smooth.base_dir = os.path.abspath('.') # define where will be
↳the root folder for the workflow
motion_correct_and_smooth.connect([
    (motion_correct, smooth, [('out_file', 'in_file')])
])
# we are connecting 'out_file' output of motion_correct to 'in_file' input of
↳smooth
motion_correct_and_smooth.run()
```

Another workflow

Another example of a simple workflow (calculate the mean of fMRI signal and subtract it). This time we'll be assigning inputs after defining the workflow.

```
calc_mean = pe.Node(interface=fsl.ImageMaths(), name="calc_mean")
calc_mean.inputs.op_string = "-Tmean"
subtract = pe.Node(interface=fsl.ImageMaths(), name="subtract")
subtract.inputs.op_string = "-sub"

demean = pe.Workflow(name="demean")
demean.base_dir = os.path.abspath('.')
demean.connect([
    (calc_mean, subtract, [('out_file', 'in_file2')])
])

demean.inputs.calc_mean.in_file = os.path.abspath('data/s1/f3.nii')
demean.inputs.subtract.in_file = os.path.abspath('data/s1/f3.nii')
demean.run()
```

Reusing workflows

The beauty of the workflows is that they are reusable. We can just import a workflow made by someone else and feed it with our data.

```
from fmri_fsl import preproc
preproc.base_dir = os.path.abspath('.')
preproc.inputs.inputs.spec.func = os.path.abspath('data/s1/f3.nii')
preproc.inputs.inputs.spec.struct = os.path.abspath('data/s1/struct.nii')
preproc.run()
```

... and we can run it again and it won't actually rerun anything because none of the parameters have changed.

```
preproc.run()
```

... and we can change a parameter and run it again. Only the dependent nodes are rerun and that too only if the input state has changed.

```
preproc.inputs.meanfuncmask.frac = 0.5
preproc.run()
```

Visualizing workflows 1

So what did we run in this precanned workflow

```
preproc.write_graph()
```

Datasink

Datasink is a special interface for copying and arranging results.

```
import nipy.interfaces.io as nio

preproc.inputs.inputs.spec.func = os.path.abspath('data/s1/f3.nii')
preproc.inputs.inputs.spec.struct = os.path.abspath('data/s1/struct.nii')
datasink = pe.Node(interface=nio.DataSink(), name='sink')
preprocess = pe.Workflow(name='preprocout')
preprocess.base_dir = os.path.abspath('.')
preprocess.connect([
    (preproc, datasink, [('meanfunc2.out_file', 'meanfunc'),
                        ('maskfunc3.out_file', 'funcruns')])
])
preprocess.run()
```

Datagrabber

Datagrabber is (surprise, surprise) an interface for collecting files from hard drive. It is very flexible and supports almost any file organisation of your data you can imagine.

```
datasource1 = nio.DataGrabber()
datasource1.inputs.template = 'data/s1/f3.nii'
datasource1.inputs.sort_filelist = True
results = datasource1.run()
print(results.outputs)
```

```

datasource2 = nio.DataGrabber()
datasource2.inputs.template = 'data/s*/f*.nii'
datasource2.inputs.sort_filelist = True
results = datasource2.run()
print(results.outputs)

datasource3 = nio.DataGrabber(infields=['run'])
datasource3.inputs.template = 'data/s1/f%d.nii'
datasource3.inputs.sort_filelist = True
datasource3.inputs.run = [3, 7]
results = datasource3.run()
print(results.outputs)

datasource4 = nio.DataGrabber(infields=['subject_id', 'run'])
datasource4.inputs.template = 'data/%s/f%d.nii'
datasource4.inputs.sort_filelist = True
datasource4.inputs.run = [3, 7]
datasource4.inputs.subject_id = ['s1', 's3']
results = datasource4.run()
print(results.outputs)

```

Iterables

Iterables is a special field of the Node class that enables to iterate all workflows/nodes connected to it over some parameters. Here we'll use it to iterate over two subjects.

```

import nipy.interfaces.utility as util
infosource = pe.Node(interface=util.IdentityInterface(fields=['subject_id']),
                      name="infosource")
infosource.iterables = ('subject_id', ['s1', 's3'])

datasource = pe.Node(nio.DataGrabber(infields=['subject_id'], outfields=['func',
↳ 'struct']), name="datasource")
datasource.inputs.template = '%s/%s.nii'
datasource.inputs.base_directory = os.path.abspath('data')
datasource.inputs.template_args = dict(func=[['subject_id', 'f3']], struct=[
↳ 'subject_id', 'struct'])
datasource.inputs.sort_filelist = True

my_workflow = pe.Workflow(name="my_workflow")
my_workflow.base_dir = os.path.abspath('.')

my_workflow.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                    (datasource, preproc, [('func', 'inputspec.func'),
                                             ('struct', 'inputspec.struct')])])

my_workflow.run()

```

and we can change a node attribute and run it again

```

smoothnode = my_workflow.get_node('preproc.smooth')
assert(str(smoothnode) == 'preproc.smooth')
smoothnode.iterables = ('fwhm', [5., 10.])
my_workflow.run()

```

Visualizing workflows 2

In the case of nested workflows, we might want to look at expanded forms of the workflow.

Example source code

You can download the full source code of this example. This same script is also included in the Nipype source distribution under the `examples` directory.

- Interfaces

ACompCor

[Link to code](#)

Anatomical compcor: for inputs and outputs, see CompCor. When the mask provided is an anatomical mask, then CompCor is equivalent to ACompCor.

Inputs:

```
[Mandatory]
realigned_file: (an existing file name)
    already realigned brain image (4D)

[Optional]
components_file: (a unicode string, nipype default value:
    components_file.txt)
    Filename to store physiological components
header_prefix: (a unicode string)
    the desired header for the output tsv file (one column). If
    undefined, will default to "CompCor"
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_files: (a list of items which are an existing file name)
    One or more mask files that determines ROI (3D). When more than one
    file is provided `merge_method` or `merge_index` must be provided
mask_index: (a long integer >= 0)
    Position of mask in `mask_files` to use - first is the default.
    mutually_exclusive: merge_method
    requires: mask_files
merge_method: ('union' or 'intersect' or 'none')
    Merge method if multiple masks are present - `union` uses voxels
    included in at least one input mask, `intersect` uses only voxels
    present in all input masks, `none` performs CompCor on each mask
    individually
    mutually_exclusive: mask_index
    requires: mask_files
num_components: (an integer (int or long), nipype default value: 6)
regress_poly_degree: (a long integer >= 1, nipype default value: 1)
```

```

        the degree polynomial to use
    use_regress_poly: (a boolean, nipy default value: True)
        use polynomial regression pre-component extraction

```

Outputs:

```

components_file: (an existing file name)
    text file containing the noise components

```

References:: None

CompCor

[Link to code](#)

Interface with core CompCor computation, used in aCompCor and tCompCor

Example

```

>>> ccinterface = CompCor()
>>> ccinterface.inputs.realigned_file = 'functional.nii'
>>> ccinterface.inputs.mask_files = 'mask.nii'
>>> ccinterface.inputs.num_components = 1
>>> ccinterface.inputs.use_regress_poly = True
>>> ccinterface.inputs.regress_poly_degree = 2

```

Inputs:

```

[Mandatory]
realigned_file: (an existing file name)
    already realigned brain image (4D)

[Optional]
components_file: (a unicode string, nipy default value:
    components_file.txt)
    Filename to store physiological components
header_prefix: (a unicode string)
    the desired header for the output tsv file (one column). If
    undefined, will default to "CompCor"
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_files: (a list of items which are an existing file name)
    One or more mask files that determines ROI (3D). When more than one
    file is provided `merge_method` or `merge_index` must be provided
mask_index: (a long integer >= 0)
    Position of mask in `mask_files` to use - first is the default.
mutually_exclusive: merge_method
    requires: mask_files
merge_method: ('union' or 'intersect' or 'none')
    Merge method if multiple masks are present - `union` uses voxels
    included in at least one input mask, `intersect` uses only voxels
    present in all input masks, `none` performs CompCor on each mask
    individually
    mutually_exclusive: mask_index
    requires: mask_files
num_components: (an integer (int or long), nipy default value: 6)
regress_poly_degree: (a long integer >= 1, nipy default value: 1)

```

```

    the degree polynomial to use
use_regress_poly: (a boolean, nipy default value: True)
    use polynomial regression pre-component extraction

```

Outputs:

```

components_file: (an existing file name)
    text file containing the noise components

```

References:: None

ComputeDVAR**S**

[Link to code](#)

Computes the DVAR**S**.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    functional data, after HMC
in_mask: (an existing file name)
    a brain mask

[Optional]
figdpi: (an integer (int or long), nipy default value: 100)
    output dpi for the plot
figformat: ('png' or 'pdf' or 'svg', nipy default value: png)
    output format for figures
figsize: (a tuple of the form: (a float, a float), nipy default
    value: (11.7, 2.3))
    output figure size
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
intensity_normalization: (a float, nipy default value: 1000.0)
    Divide value in each voxel at each timepoint by the median
    calculated across all voxels and timepoints within the mask (if
    specified) and then multiply by the value specified by this parameter.
    By using the default (1000) output DVARS will be expressed in x10 %
    BOLD units compatible with Power et al.2012. Set this to 0 to
    disable intensity normalization altogether.
remove_zerovariance: (a boolean, nipy default value: True)
    remove voxels with zero variance
save_all: (a boolean, nipy default value: False)
    output all DVARS
save_nstd: (a boolean, nipy default value: False)
    save non-standardized DVARS
save_plot: (a boolean, nipy default value: False)
    write DVARS plot
save_std: (a boolean, nipy default value: True)
    save standardized DVARS
save_vxstd: (a boolean, nipy default value: False)
    save voxel-wise standardized DVARS
series_tr: (a float)
    repetition time in sec.

```

Outputs:

```
avg_nstd: (a float)
avg_std: (a float)
avg_vxstd: (a float)
fig_nstd: (an existing file name)
        output DVARS plot
fig_std: (an existing file name)
        output DVARS plot
fig_vxstd: (an existing file name)
        output DVARS plot
out_all: (an existing file name)
        output text file
out_nstd: (an existing file name)
        output text file
out_std: (an existing file name)
        output text file
out_vxstd: (an existing file name)
        output text file
```

References:: None None

FrameworkDisplacement

[Link to code](#)

Calculate the FD (framework displacement) as in *[Power2012]*. This implementation reproduces the calculation in `fsl_motion_outliers`

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        motion parameters
parameter_source: ('FSL' or 'AFNI' or 'SPM' or 'FSFAST' or 'NIPY')
        Source of movement parameters

[Optional]
figdpi: (an integer (int or long), nipy default value: 100)
        output dpi for the FD plot
figsize: (a tuple of the form: (a float, a float), nipy default
        value: (11.7, 2.3))
        output figure size
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
normalize: (a boolean, nipy default value: False)
        calculate FD in mm/s
out_figure: (a file name, nipy default value: fd_power_2012.pdf)
        output figure name
out_file: (a file name, nipy default value: fd_power_2012.txt)
        output file name
radius: (a float, nipy default value: 50)
        radius in mm to calculate angular FDs, 50mm is the default since it
        is used in Power et al. 2012
save_plot: (a boolean, nipy default value: False)
        write FD plot
series_tr: (a float)
        repetition time in sec.
```

Outputs:


```

fd_average: (a float)
    average FD
out_figure: (a file name)
    output image file
out_file: (a file name)
    calculated FD per timestep

```

References:: None

NonSteadyStateDetector

[Link to code](#)

Returns the number of non-steady state volumes detected at the beginning of the scan.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    4D NIFTI EPI file

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run

```

Outputs:

```

n_volumes_to_discard: (an integer (int or long))
    Number of non-steady state volumes detected in the beginning of the
    scan.

```

TCompCor

[Link to code](#)

Interface for tCompCor. Computes a ROI mask based on variance of voxels.

Example

```

>>> ccinterface = TCompCor()
>>> ccinterface.inputs.realigned_file = 'functional.nii'
>>> ccinterface.inputs.mask_files = 'mask.nii'
>>> ccinterface.inputs.num_components = 1
>>> ccinterface.inputs.use_regress_poly = True
>>> ccinterface.inputs.regress_poly_degree = 2
>>> ccinterface.inputs.percentile_threshold = .03

```

Inputs:

```

[Mandatory]
realigned_file: (an existing file name)
    already realigned brain image (4D)

[Optional]
components_file: (a unicode string, nipy default value:
    components_file.txt)
    Filename to store physiological components
header_prefix: (a unicode string)

```

```

        the desired header for the output tsv file (one column). If
        undefined, will default to "CompCor"
    ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
    mask_files: (a list of items which are an existing file name)
        One or more mask files that determines ROI (3D). When more than one
        file is provided `merge_method` or `merge_index` must be provided
    mask_index: (a long integer >= 0)
        Position of mask in `mask_files` to use - first is the default.
    mutually_exclusive: merge_method
    requires: mask_files
    merge_method: ('union' or 'intersect' or 'none')
        Merge method if multiple masks are present - `union` uses voxels
        included in at least one input mask, `intersect` uses only voxels
        present in all input masks, `none` performs CompCor on each mask
        individually
    mutually_exclusive: mask_index
    requires: mask_files
    num_components: (an integer (int or long), nipy default value: 6)
    percentile_threshold: (0.0 < a floating point number < 1.0, nipy
        default value: 0.02)
        the percentile used to select highest-variance voxels, represented
        by a number between 0 and 1, exclusive. By default, this value is
        set to .02. That is, the 2% of voxels with the highest variance are
        used.
    regress_poly_degree: (a long integer >= 1, nipy default value: 1)
        the degree polynomial to use
    use_regress_poly: (a boolean, nipy default value: True)
        use polynomial regression pre-component extraction

```

Outputs:

```

components_file: (an existing file name)
    text file containing the noise components
high_variance_masks: (a list of items which are an existing file
    name)
    voxels exceeding the variance threshold

```

References:: None

TSNR

[Link to code](#)

Computes the time-course SNR for a time series

Typically you want to run this on a realigned time-series.

Example

```

>>> tsnr = TSNR()
>>> tsnr.inputs.in_file = 'functional.nii'
>>> res = tsnr.run()

```

Inputs:

```

[Mandatory]
in_file: (a list of items which are an existing file name)

```

```

    realigned 4D file or a list of 3D files

[Optional]
detrended_file: (a file name, nipy default value: detrend.nii.gz)
    input file after detrending
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mean_file: (a file name, nipy default value: mean.nii.gz)
    output mean file
regress_poly: (a long integer >= 1)
    Remove polynomials
stddev_file: (a file name, nipy default value: stdev.nii.gz)
    output tSNR file
tsnr_file: (a file name, nipy default value: tsnr.nii.gz)
    output tSNR file

```

Outputs:

```

detrended_file: (a file name)
    detrended input file
mean_file: (an existing file name)
    mean image file
stddev_file: (an existing file name)
    std dev image file
tsnr_file: (an existing file name)
    tsnr image file

```

combine_mask_files()[Link to code](#)

Combines input mask files into a single nibabel image

A helper function for CompCor

mask_files: a list one or more binary mask files

mask_method: enum ('union', 'intersect', 'none') determines how to combine masks

mask_index: an integer determines which file to return (mutually exclusive with mask_method)

returns: a list of nibabel images

compute_dvars()[Link to code](#)

Compute the DVARS (D referring to temporal derivative of timecourses, VARS referring to RMS variance over voxels) [\[Power2012\]](#).

Particularly, the *standardized* DVARS [\[Nichols2013\]](#) are computed.

Note: Implementation details

Uses the implementation of the [Yule-Walker equations](#) from `nitime` for the AR (auto-regressive) filtering of the fMRI signal.

param numpy.ndarray func functional data, after head-motion-correction.

param numpy.ndarray mask a 3D mask of the brain

param bool output_all write out all dvars

param str out_file a path to which the standardized dvars should be saved.

return the standardized DVARS

compute_noise_components()

[Link to code](#)

Compute the noise components from the imgseries for each mask

imgseries: a nibabel img mask_images: a list of nibabel images degree: order of polynomial used to remove trends from the timeseries num_components: number of noise components to return

returns:

components: a numpy array

is_outlier()

[Link to code](#)

Returns a boolean array with True if points are outliers and False otherwise.

param nparray points an numobservations by numdimensions numpy array of observations

param float thresh the modified z-score to use as a threshold. Observations with a modified z-score (based on the median absolute deviation) greater than this value will be classified as outliers.

return A boolean mask, of size numobservations-length array.

Note: References

Boris Iglewicz and David Hoaglin (1993), “Volume 16: How to Detect and Handle Outliers”, The ASQC Basic References in Quality Control: Statistical Techniques, Edward F. Mykytka, Ph.D., Editor.

plot_confound()

[Link to code](#)

A helper function to plot FMRI (functional MRI) confounds.

regress_poly()

[Link to code](#)

Returns data with degree polynomial regressed out.

param bool remove_mean whether or not demean data (i.e. degree 0),

param int axis numpy array axes along which regression is performed

ICC

[Link to code](#)

Calculates Interclass Correlation Coefficient (3,1) as defined in P. E. Shrout & Joseph L. Fleiss (1979). “Intra-class Correlations: Uses in Assessing Rater Reliability”. Psychological Bulletin 86 (2): 420-428. This particular implementation is aimed at reliability (test-retest) studies.

Inputs:

```
[Mandatory]
mask: (an existing file name)
subjects_sessions: (a list of items which are a list of items which
                    are an existing file name)
                    n subjects m sessions 3D stat files

[Optional]
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
```

Outputs:

```
icc_map: (an existing file name)
session_var_map: (an existing file name)
                variance between sessions
subject_var_map: (an existing file name)
                variance between subjects
```

ICC_rep_anova()

[Link to code](#)

the data Y are entered as a ‘table’ ie subjects are in rows and repeated measures in columns

One Sample Repeated measure ANOVA

$Y = XB + E$ with $X = [\text{FaTor} / \text{Subjects}]$

ComputeMeshWarp

[Link to code](#)

Calculates a the vertex-wise warping to get surface2 from surface1. It also reports the average distance of vertices, using the norm specified as input.

Example:

```
import nipy.algorithms.mesh as m
dist = m.ComputeMeshWarp()
dist.inputs.surface1 = 'surf1.vtk'
dist.inputs.surface2 = 'surf2.vtk'
res = dist.run()
```

Inputs:

```
[Mandatory]
surface1: (an existing file name)
    Reference surface (vtk format) to which compute distance.
surface2: (an existing file name)
    Test surface (vtk format) from which compute distance.

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
metric: ('euclidean' or 'sqeuclidean', nipy default value:
    euclidean)
    norm used to report distance
out_file: (a file name, nipy default value: distance.npy)
    numpy file keeping computed distances and weights
out_warp: (a file name, nipy default value: surfwarp.vtk)
    vtk file based on surface1 and warpings mapping it to surface2
weighting: ('none' or 'area', nipy default value: none)
    "none": no weighting is performed, surface": edge distance is
    weighted by the corresponding surface area
```

Outputs:

```
distance: (a float)
    computed distance
out_file: (an existing file name)
    numpy file keeping computed distances and weights
out_warp: (an existing file name)
    vtk file with the vertex-wise mapping of surface1 to surface2
```

MeshWarpMaths

[Link to code](#)

Performs the most basic mathematical operations on the warping field defined at each vertex of the input surface. A surface with scalar or vector data can be used as operator for non-uniform operations.

Example:

```
import nipyype.algorithms.mesh as m
mmath = m.MeshWarpMaths()
mmath.inputs.in_surf = 'surf1.vtk'
mmath.inputs.operator = 'surf2.vtk'
mmath.inputs.operation = 'mul'
res = mmath.run()
```

Inputs:

```
[Mandatory]
in_surf: (an existing file name)
    Input surface in vtk format, with associated warp field as point
    data (ie. from ComputeMeshWarp)
operator: (a float or a tuple of the form: (a float, a float, a
    float) or an existing file name)
    image, float or tuple of floats to act as operator

[Optional]
float_trait: (a float or a tuple of the form: (a float, a float, a
    float))
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
operation: ('sum' or 'sub' or 'mul' or 'div', nipyype default value:
    sum)
    operation to be performed
out_file: (a file name, nipyype default value: warped_surf.vtk)
    vtk with surface warped
out_warp: (a file name, nipyype default value: warp_maths.vtk)
    vtk file based on in_surf and warpings mapping it to out_file
```

Outputs:

```
out_file: (an existing file name)
    vtk with surface warped
out_warp: (an existing file name)
    vtk file with the vertex-wise mapping of surface1 to surface2
```

P2PDistance

[Link to code](#)

Calculates a point-to-point (p2p) distance between two corresponding VTK-readable meshes or contours.

A point-to-point correspondence between nodes is required

Deprecated since version 1.0-dev: Use ComputeMeshWarp instead.

Inputs:

```
[Mandatory]
surface1: (an existing file name)
    Reference surface (vtk format) to which compute distance.
surface2: (an existing file name)
    Test surface (vtk format) from which compute distance.

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
metric: ('euclidean' or 'sqeuclidean', nipy default value:
    euclidean)
    norm used to report distance
out_file: (a file name, nipy default value: distance.npy)
    numpy file keeping computed distances and weights
out_warp: (a file name, nipy default value: surfwarp.vtk)
    vtk file based on surface1 and warplings mapping it to surface2
weighting: ('none' or 'area', nipy default value: none)
    "none": no weighting is performed, surface": edge distance is
    weighted by the corresponding surface area
```

Outputs:

```
distance: (a float)
    computed distance
out_file: (an existing file name)
    numpy file keeping computed distances and weights
out_warp: (an existing file name)
    vtk file with the vertex-wise mapping of surface1 to surface2
```

TVTKBaseInterface

[Link to code](#)

A base class for interfaces using VTK

Inputs:

```
[Mandatory]

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

Outputs:

None

WarpPoints

[Link to code](#)

Applies a displacement field to a point set given in vtk format. Any discrete deformation field, given in physical coordinates and which volume covers the extent of the vtk point set, is a valid warp file. FSL interfaces are

compatible, for instance any field computed with `nipyype.interfaces.fsl.utils.ConvertWarp`.
Example:

```
from nipyype.algorithms.mesh import WarpPoints
wp = WarpPoints()
wp.inputs.points = 'surfl.vtk'
wp.inputs.warp = 'warpfield.nii'
res = wp.run()
```

Inputs:

```
[Mandatory]
interp: ('cubic' or 'nearest' or 'linear', nipyype default value:
        cubic)
        interpolation
points: (an existing file name)
        file containing the point set
warp: (an existing file name)
       dense deformation field to be applied

[Optional]
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_points: (a file name)
            the warped point set
```

Outputs:

```
out_points: (a file name)
            the warped point set
```

Distance

[Link to code](#)

Calculates distance between two volumes.

Inputs:

```
[Mandatory]
volume1: (an existing file name)
        Has to have the same dimensions as volume2.
volume2: (an existing file name)
        Has to have the same dimensions as volume1.

[Optional]
ignore_exception: (a boolean, nipype default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
mask_volume: (an existing file name)
        calculate overlap only within this mask.
method: ('eucl_min' or 'eucl_cog' or 'eucl_mean' or 'eucl_wmean' or
        'eucl_max', nipype default value: eucl_min)
        "eucl_min": Euclidean distance between two closest points
        "eucl_cog": mean Euclidian distance between the Center of Gravity of
        volume1 and CoGs of volume2 "eucl_mean": mean Euclidian minimum
        distance of all volume2 voxels to volume1 "eucl_wmean": mean
        Euclidian minimum distance of all volume2 voxels to volume1 weighted
        by their values "eucl_max": maximum over minimum Euclidian distances
        of all volume2 voxels to volume1 (also known as the Hausdorff
        distance)
```

Outputs:

```
distance: (a float)
histogram: (a file name)
point1: (an array with shape (3,))
point2: (an array with shape (3,))
```

ErrorMap

[Link to code](#)

Calculates the error (distance) map between two input volumes.

Example

```
>>> errormap = ErrorMap()
>>> errormap.inputs.in_ref = 'cont1.nii'
>>> errormap.inputs.in_tst = 'cont2.nii'
>>> res = errormap.run()
```

Inputs:

```
[Mandatory]
in_ref: (an existing file name)
        Reference image. Requires the same dimensions as in_tst.
in_tst: (an existing file name)
        Test image. Requires the same dimensions as in_ref.
metric: ('sqeuclidean' or 'euclidean', nipy default value:
        sqeuclidean)
        error map metric (as implemented in scipy cdist)

[Optional]
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
mask: (an existing file name)
        calculate overlap only within this mask.
out_map: (a file name)
        Name for the output file
```

Outputs:

```
distance: (a float)
        Average distance between volume 1 and 2
out_map: (an existing file name)
        resulting error map
```

FuzzyOverlap

[Link to code](#)

Calculates various overlap measures between two maps, using the fuzzy definition proposed in: Crum et al., Generalized Overlap Measures for Evaluation and Validation in Medical Image Analysis, IEEE Trans. Med. Ima. 25(11),pp 1451-1461, Nov. 2006.

in_ref and in_tst are lists of 2/3D images, each element on the list containing one volume fraction map of a class in a fuzzy partition of the domain.

Example

```
>>> overlap = FuzzyOverlap()
>>> overlap.inputs.in_ref = [ 'ref_class0.nii', 'ref_class1.nii' ]
>>> overlap.inputs.in_tst = [ 'tst_class0.nii', 'tst_class1.nii' ]
>>> overlap.inputs.weighting = 'volume'
>>> res = overlap.run()
```

Inputs:

```
[Mandatory]
in_ref: (a list of items which are an existing file name)
        Reference image. Requires the same dimensions as in_tst.
in_tst: (a list of items which are an existing file name)
        Test image. Requires the same dimensions as in_ref.

[Optional]
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
out_file: (a file name, nipy default value: diff.nii)
        alternative name for resulting difference-map
weighting: ('none' or 'volume' or 'squared_vol', nipy default
        value: none)
        'none': no class-overlap weighting is performed. 'volume': computed
        class-overlaps are weighted by class volume 'squared_vol': computed
        class-overlaps are weighted by the squared volume of the class
```

Outputs:

```
class_fdi: (a list of items which are a float)
        Array containing the fDIs of each computed class
class_fji: (a list of items which are a float)
        Array containing the fJIs of each computed class
dice: (a float)
        Fuzzy Dice Index (fDI), all the classes
diff_file: (an existing file name)
        resulting difference-map of all classes, using the chosen weighting
jaccard: (a float)
        Fuzzy Jaccard Index (fJI), all the classes
```

Overlap

[Link to code](#)

Calculates Dice and Jaccard's overlap measures between two ROI maps. The interface is backwards compatible with the former version in which only binary files were accepted.

The averaged values of overlap indices can be weighted. Volumes now can be reported in mm^3 , although they are given in voxels to keep backwards compatibility.

Example

```
>>> overlap = Overlap()
>>> overlap.inputs.volume1 = 'cont1.nii'
>>> overlap.inputs.volume2 = 'cont2.nii'
>>> res = overlap.run()
```

Inputs:

```
[Mandatory]
bg_overlap: (a boolean, nipy default value: False)
        consider zeros as a label
vol_units: ('voxel' or 'mm', nipy default value: voxel)
        units for volumes
volume1: (an existing file name)
        Has to have the same dimensions as volume2.
```

```
volume2: (an existing file name)
    Has to have the same dimensions as volume1.

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_volume: (an existing file name)
    calculate overlap only within this mask.
out_file: (a file name, nipy default value: diff.nii)
weighting: ('none' or 'volume' or 'squared_vol', nipy default
    value: none)
    'none': no class-overlap weighting is performed. 'volume': computed
    class-overlaps are weighted by class volume 'squared_vol': computed
    class-overlaps are weighted by the squared volume of the class
```

Outputs:

```
dice: (a float)
    averaged dice index
diff_file: (an existing file name)
    error map of differences
jaccard: (a float)
    averaged jaccard index
labels: (a list of items which are an integer (int or long))
    detected labels
roi_di: (a list of items which are a float)
    the Dice index (DI) per ROI
roi_ji: (a list of items which are a float)
    the Jaccard index (JI) per ROI
roi_voldiff: (a list of items which are a float)
    volume differences of ROIs
volume_difference: (a float)
    averaged volume difference
```

Similarity

[Link to code](#)

Calculates similarity between two 3D or 4D volumes. Both volumes have to be in the same coordinate system, same space within that coordinate system and with the same voxel dimensions.

Note: This interface is an extension of `nipy.interfaces.nipy.utils.Similarity` to support 4D files. Requires `nipy`

Example

```
>>> from nipy.algorithms.metrics import Similarity
>>> similarity = Similarity()
>>> similarity.inputs.volume1 = 'rc1s1.nii'
>>> similarity.inputs.volume2 = 'rc1s2.nii'
>>> similarity.inputs.mask1 = 'mask.nii'
>>> similarity.inputs.mask2 = 'mask.nii'
>>> similarity.inputs.metric = 'cr'
>>> res = similarity.run()
```

Inputs:

```
[Mandatory]
volume1: (an existing file name)
        3D/4D volume
volume2: (an existing file name)
        3D/4D volume

[Optional]
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask1: (an existing file name)
        3D volume
mask2: (an existing file name)
        3D volume
metric: ('cc' or 'cr' or 'crl1' or 'mi' or 'nmi' or 'slr' or a
         callable value, nipy default value: None)
         str or callable
         Cost-function for assessing image similarity. If a string,
         one of 'cc': correlation coefficient, 'cr': correlation
         ratio, 'crl1': L1-norm based correlation ratio, 'mi': mutual
         information, 'nmi': normalized mutual information, 'slr':
         supervised log-likelihood ratio. If a callable, it should
         take a two-dimensional array representing the image joint
         histogram as an input and return a float.
```

Outputs:

```
similarity: (a list of items which are a float)
```


AddCSVColumn

[Link to code](#)

Short interface to add an extra column and field to a text file

Example

```
>>> from nipy.algorithms import misc
>>> addcol = misc.AddCSVColumn()
>>> addcol.inputs.in_file = 'degree.csv'
>>> addcol.inputs.extra_column_heading = 'group'
>>> addcol.inputs.extra_field = 'male'
>>> addcol.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Input comma-separated value (CSV) files

[Optional]
extra_column_heading: (a unicode string)
        New heading to add for the added field.
extra_field: (a unicode string)
        New field to add to each row. This is useful for saving the group or
        subject ID in the file.
out_file: (a file name, nipy default value: extra_heading.csv)
        Output filename for merged CSV file
```

Outputs:

```
csv_file: (a file name)
        Output CSV file containing columns
```

AddCSVRow

[Link to code](#)

Simple interface to add an extra row to a csv file

Note: Requires `pandas`

Warning: Multi-platform thread-safe execution is possible with `lockfile`. Please recall that (1) this module is alpha software; and (2) it should be installed for thread-safe writing. If `lockfile` is not installed, then the interface is not thread-safe.

Example

```
>>> from nipyte.algorithms import misc
>>> addrow = misc.AddCSVRow()
>>> addrow.inputs.in_file = 'scores.csv'
>>> addrow.inputs.si = 0.74
>>> addrow.inputs.di = 0.93
>>> addrow.inputs.subject_id = 'S400'
>>> addrow.inputs.list_of_values = [ 0.4, 0.7, 0.3 ]
>>> addrow.run()
```

Inputs:

```
[Mandatory]
in_file: (a file name)
        Input comma-separated value (CSV) files

[Optional]
_outputs: (a dictionary with keys which are any value and with values
          which are any value, nipyte default value: {})
ignore_exception: (a boolean, nipyte default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
```

Outputs:

```
csv_file: (a file name)
          Output CSV file containing rows
```

AddNoise

[Link to code](#)

Corrupts with noise the input image

Example

```
>>> from nipyte.algorithms.misc import AddNoise
>>> noise = AddNoise()
>>> noise.inputs.in_file = 'T1.nii'
>>> noise.inputs.in_mask = 'mask.nii'
>>> noise.snr = 30.0
>>> noise.run()
```

Inputs:

```
[Mandatory]
bg_dist: ('normal' or 'rayleigh', nipyype default value: normal)
        desired noise distribution, currently only normal is implemented
dist: ('normal' or 'rician', nipyype default value: normal)
      desired noise distribution
in_file: (an existing file name)
        input image that will be corrupted with noise

[Optional]
in_mask: (an existing file name)
        input mask, voxels outside this mask will be considered background
out_file: (a file name)
          desired output filename
snr: (a float, nipyype default value: 10.0)
     desired output SNR in dB
```

Outputs:

```
out_file: (an existing file name)
          corrupted image
```

CalculateNormalizedMoments

[Link to code](#)

Calculates moments of timeseries.

Example

```
>>> from nipyype.algorithms import misc
>>> skew = misc.CalculateNormalizedMoments()
>>> skew.inputs.moment = 3
>>> skew.inputs.timeseries_file = 'timeseries.txt'
>>> skew.run()
```

Inputs:

```
[Mandatory]
moment: (an integer (int or long))
        Define which moment should be calculated, 3 for skewness, 4 for
        kurtosis.
timeseries_file: (an existing file name)
                 Text file with timeseries in columns and timepoints in rows,
                 whitespace separated

[Optional]
```

Outputs:

```
moments: (a list of items which are a float)
          Moments
```

CreateNifti

[Link to code](#)

Creates a nifti volume

Inputs:

```
[Mandatory]
data_file: (an existing file name)
           ANALYZE img file
header_file: (an existing file name)
             corresponding ANALYZE hdr file

[Optional]
affine: (an array)
        affine transformation array
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
```

Outputs:

```
nifti_file: (an existing file name)
```

Distance

[Link to code](#)

Calculates distance between two volumes.

Deprecated since version 0.10.0: Use `nipyype.algorithms.metrics.Distance` instead.

Inputs:

```
[Mandatory]
volume1: (an existing file name)
          Has to have the same dimensions as volume2.
volume2: (an existing file name)
          Has to have the same dimensions as volume1.

[Optional]
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask_volume: (an existing file name)
              calculate overlap only within this mask.
method: ('eucl_min' or 'eucl_cog' or 'eucl_mean' or 'eucl_wmean' or
         'eucl_max', nipyype default value: eucl_min)
         "eucl_min": Euclidean distance between two closest points
         "eucl_cog": mean Euclidian distance between the Center of Gravity of
         volume1 and CoGs of volume2 "eucl_mean": mean Euclidian minimum
         distance of all volume2 voxels to volume1 "eucl_wmean": mean
         Euclidian minimum distance of all volume2 voxels to volume1 weighted
         by their values "eucl_max": maximum over minimum Euclidian distances
         of all volume2 voxels to volume1 (also known as the Hausdorff
         distance)
```

Outputs:

```
distance: (a float)
histogram: (a file name)
point1: (an array with shape (3,))
point2: (an array with shape (3,))
```

FuzzyOverlap

[Link to code](#)

Calculates various overlap measures between two maps, using a fuzzy definition.

Deprecated since version 0.10.0: Use `nipy.algorithms.metrics.FuzzyOverlap` instead.

Inputs:

```
[Mandatory]
in_ref: (a list of items which are an existing file name)
        Reference image. Requires the same dimensions as in_tst.
in_tst: (a list of items which are an existing file name)
        Test image. Requires the same dimensions as in_ref.

[Optional]
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name, nipy default value: diff.nii)
          alternative name for resulting difference-map
weighting: ('none' or 'volume' or 'squared_vol', nipy default
            value: none)
            'none': no class-overlap weighting is performed. 'volume': computed
            class-overlaps are weighted by class volume 'squared_vol': computed
            class-overlaps are weighted by the squared volume of the class
```

Outputs:

```
class_fdi: (a list of items which are a float)
           Array containing the fDIs of each computed class
class_fji: (a list of items which are a float)
           Array containing the fJIs of each computed class
dice: (a float)
       Fuzzy Dice Index (fDI), all the classes
diff_file: (an existing file name)
           resulting difference-map of all classes, using the chosen weighting
jaccard: (a float)
         Fuzzy Jaccard Index (fJI), all the classes
```

Gunzip

[Link to code](#)

Gunzip wrapper

Inputs:

```
[Mandatory]
in_file: (an existing file name)

[Optional]
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
```

Outputs:

```
out_file: (an existing file name)
```

Matlab2CSV

[Link to code](#)

Simple interface to save the components of a MATLAB .mat file as a text file with comma-separated values (CSVs).

CSV files are easily loaded in R, for use in statistical processing. For further information, see cran.r-project.org/doc/manuals/R-data.pdf

Example

```
>>> from nipyre.algorithms import misc
>>> mat2csv = misc.Matlab2CSV()
>>> mat2csv.inputs.in_file = 'cmatrix.mat'
>>> mat2csv.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         Input MATLAB .mat file

[Optional]
reshape_matrix: (a boolean, nipyre default value: True)
                 The output of this interface is meant for R, so matrices will be
                 reshaped to vectors by default.
```

Outputs:

```
csv_files: (a list of items which are a file name)
```

MergeCSVFiles

[Link to code](#)

This interface is designed to facilitate data loading in the R environment. It takes input CSV files and merges them into a single CSV file. If provided, it will also incorporate column heading names into the resulting CSV file.

CSV files are easily loaded in R, for use in statistical processing. For further information, see cran.r-project.org/doc/manuals/R-data.pdf

Example

```
>>> from nipyre.algorithms import misc
>>> mat2csv = misc.MergeCSVFiles()
>>> mat2csv.inputs.in_files = ['degree.mat', 'clustering.mat']
>>> mat2csv.inputs.column_headings = ['degree', 'clustering']
>>> mat2csv.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
          Input comma-separated value (CSV) files

[Optional]
column_headings: (a list of items which are a unicode string)
```

```

        List of column headings to save in merged CSV file (must be equal to
        number of input files). If left undefined, these will be pulled from
        the input filenames.
    extra_column_heading: (a unicode string)
        New heading to add for the added field.
    extra_field: (a unicode string)
        New field to add to each row. This is useful for saving the group or
        subject ID in the file.
    out_file: (a file name, nipy default value: merged.csv)
        Output filename for merged CSV file
    row_heading_title: (a unicode string, nipy default value: label)
        Column heading for the row headings added
    row_headings: (a list of items which are a unicode string)
        List of row headings to save in merged CSV file (must be equal to
        number of rows in the input files).

```

Outputs:

```

csv_file: (a file name)
    Output CSV file containing columns

```

MergeROIs

[Link to code](#)

Splits a 3D image in small chunks to enable parallel processing. ROIs keep time series structure in 4D images.

Example

```

>>> from nipy.algorithms import misc
>>> rois = misc.MergeROIs()
>>> rois.inputs.in_files = ['roi%02d.nii' % i for i in range(1, 6)]
>>> rois.inputs.in_reference = 'mask.nii'
>>> rois.inputs.in_index = ['roi%02d_idx.npz' % i for i in range(1, 6)]
>>> rois.run()

```

Inputs:

```

[Mandatory]

[Optional]
in_files: (a list of items which are an existing file name)
in_index: (a list of items which are an existing file name)
    array keeping original locations
in_reference: (an existing file name)
    reference file

```

Outputs:

```

merged_file: (an existing file name)
    the recomposed file

```

ModifyAffine

[Link to code](#)

Left multiplies the affine matrix with a specified values. Saves the volume as a nifti file.

Inputs:

```
[Mandatory]
volumes: (a list of items which are an existing file name)
          volumes which affine matrices will be modified

[Optional]
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
transformation_matrix: (an array with shape (4, 4), nipy default
                        value: (<bound method AbstractArray.copy_default_value of
                        <traits.trait_numeric.Array object at 0x7f07699264e0>>, (array([[
                        1., 0., 0., 0.], [ 0., 1., 0., 0.], [ 0., 0.,
                        1., 0.], [ 0., 0., 0., 1.])),), None))
                        transformation matrix that will be left multiplied by the affine
                        matrix
```

Outputs:

```
transformed_volumes: (a list of items which are a file name)
```

NormalizeProbabilityMapSet

[Link to code](#)

Returns the input tissue probability maps (tpms, aka volume fractions) normalized to sum up 1.0 at each voxel within the mask.

Note: Please recall this is not a spatial normalization algorithm

Example

```
>>> from nipy.algorithms import misc
>>> normalize = misc.NormalizeProbabilityMapSet()
>>> normalize.inputs.in_files = [ 'tpm_00.nii.gz', 'tpm_01.nii.gz', 'tpm_02.nii.gz
↪' ]
>>> normalize.inputs.in_mask = 'tpms_msk.nii.gz'
>>> normalize.run()
```

Inputs:

```
[Mandatory]

[Optional]
in_files: (a list of items which are an existing file name)
in_mask: (an existing file name)
          Masked voxels must sum up 1.0, 0.0 otherwise.
```

Outputs:

```
out_files: (a list of items which are an existing file name)
            normalized maps
```


Overlap

[Link to code](#)

Calculates various overlap measures between two maps.

Deprecated since version 0.10.0: Use `nipy.algorithms.metrics.Overlap` instead.

Inputs:

```
[Mandatory]
bg_overlap: (a boolean, nipy default value: False)
             consider zeros as a label
vol_units: ('voxel' or 'mm', nipy default value: voxel)
            units for volumes
volume1: (an existing file name)
          Has to have the same dimensions as volume2.
volume2: (an existing file name)
          Has to have the same dimensions as volume1.

[Optional]
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask_volume: (an existing file name)
              calculate overlap only within this mask.
out_file: (a file name, nipy default value: diff.nii)
weighting: ('none' or 'volume' or 'squared_vol', nipy default
            value: none)
            'none': no class-overlap weighting is performed. 'volume': computed
            class-overlaps are weighted by class volume 'squared_vol': computed
            class-overlaps are weighted by the squared volume of the class
```

Outputs:

```
dice: (a float)
      averaged dice index
diff_file: (an existing file name)
           error map of differences
jaccard: (a float)
         averaged jaccard index
labels: (a list of items which are an integer (int or long))
        detected labels
roi_di: (a list of items which are a float)
        the Dice index (DI) per ROI
roi_ji: (a list of items which are a float)
        the Jaccard index (JI) per ROI
roi_voldiff: (a list of items which are a float)
             volume differences of ROIs
volume_difference: (a float)
                  averaged volume difference
```

PickAtlas

[Link to code](#)

Returns ROI masks given an atlas and a list of labels. Supports dilation and left right masking (assuming the atlas is properly aligned).

Inputs:

```
[Mandatory]
atlas: (an existing file name)
    Location of the atlas that will be used.
labels: (an integer (int or long) or a list of items which are an
    integer (int or long))
    Labels of regions that will be included in the mask. Must be
    compatible with the atlas used.

[Optional]
dilation_size: (an integer (int or long), nipy default value: 0)
    Defines how much the mask will be dilated (expanded in 3D).
hemi: ('both' or 'left' or 'right', nipy default value: both)
    Restrict the mask to only one hemisphere: left or right
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_file: (a file name)
    Where to store the output mask.
```

Outputs:

```
mask_file: (an existing file name)
    output mask file
```

SimpleThreshold

[Link to code](#)

Applies a threshold to input volumes

Inputs:

```
[Mandatory]
threshold: (a float)
    volumes to be thresholdedeverything below this value will be set to
    zero
volumes: (a list of items which are an existing file name)
    volumes to be thresholded

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

Outputs:

```
thresholded_volumes: (a list of items which are an existing file
    name)
    thresholded volumes
```

SplitROIs

[Link to code](#)

Splits a 3D image in small chunks to enable parallel processing. ROIs keep time series structure in 4D images.

Example

```
>>> from nipyype.algorithms import misc
>>> rois = misc.SplitROIs()
>>> rois.inputs.in_file = 'diffusion.nii'
>>> rois.inputs.in_mask = 'mask.nii'
>>> rois.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         file to be splitted

[Optional]
in_mask: (an existing file name)
         only process files inside mask
roi_size: (a tuple of the form: (an integer (int or long), an integer
                                (int or long), an integer (int or long)))
         desired ROI size
```

Outputs:

```
out_files: (a list of items which are an existing file name)
           the resulting ROIs
out_index: (a list of items which are an existing file name)
           arrays keeping original locations
out_masks: (a list of items which are an existing file name)
           a mask indicating valid values
```

TSNR

[Link to code](#)

Deprecated since version 0.12.1: Use `nipyype.algorithms.confounds.TSNR` instead

Inputs:

```
[Mandatory]
in_file: (a list of items which are an existing file name)
         realigned 4D file or a list of 3D files

[Optional]
detrended_file: (a file name, nipyype default value: detrend.nii.gz)
                input file after detrending
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mean_file: (a file name, nipyype default value: mean.nii.gz)
            output mean file
regress_poly: (a long integer >= 1)
              Remove polynomials
stddev_file: (a file name, nipyype default value: stddev.nii.gz)
             output tSNR file
tsnr_file: (a file name, nipyype default value: tsnr.nii.gz)
            output tSNR file
```

Outputs:

```
detrended_file: (a file name)
                 detrended input file
mean_file: (an existing file name)
            mean image file
stddev_file: (an existing file name)
              std dev image file
tsnr_file: (an existing file name)
            tsnr image file
```

calc_moments()

[Link to code](#)

Returns nth moment (3 for skewness, 4 for kurtosis) of timeseries (list of values; one per timeseries).

Keyword arguments: timeseries_file – text file with white space separated timepoints in rows

makefmtlist()

[Link to code](#)

maketypelist()

[Link to code](#)

matlab2csv()

[Link to code](#)

merge_csvs()

[Link to code](#)

merge_rois()

[Link to code](#)

Re-builds an image resulting from a parallelized processing

normalize_tpms()

[Link to code](#)

Returns the input tissue probability maps (tpms, aka volume fractions) normalized to sum up 1.0 at each voxel within the mask.

remove_identical_paths()

[Link to code](#)

replaceext()

[Link to code](#)

split_rois()

[Link to code](#)

Splits an image in ROIs for parallel processing

SpecifyModel

[Link to code](#)

Makes a model specification compatible with spm/fsl designers.

The `subject_info` field should contain paradigm information in the form of a Bunch or a list of Bunch. The Bunch should contain the following information:

```
[Mandatory]
- conditions : list of names
- onsets : lists of onsets corresponding to each condition
- durations : lists of durations corresponding to each condition. Should be
left to a single 0 if all events are being modelled as impulses.

[Optional]
- regressor_names : list of str
    list of names corresponding to each column. Should be None if
    automatically assigned.
- regressors : list of lists
    values for each regressor - must correspond to the number of
    volumes in the functional run
- amplitudes : lists of amplitudes for each event. This will be ignored by
SPM's Level1Design.

The following two (tmod, pmod) will be ignored by any Level1Design class
other than SPM:

- tmod : lists of conditions that should be temporally modulated. Should
default to None if not being used.
- pmod : list of Bunch corresponding to conditions
    - name : name of parametric modulator
    - param : values of the modulator
    - poly : degree of modulation
```

Alternatively, you can provide information through event files.

The event files have to be in 1, 2 or 3 column format with the columns corresponding to Onsets, Durations and Amplitudes and they have to have the name `event_name.runXXX...` e.g.: `Words.run001.txt`. The `event_name` part will be used to create the condition names.

Examples

```
>>> from nipyype.algorithms import modelgen
>>> from nipyype.interfaces.base import Bunch
>>> s = modelgen.SpecifyModel()
>>> s.inputs.input_units = 'secs'
>>> s.inputs.functional_runs = ['functional2.nii', 'functional3.nii']
>>> s.inputs.time_repetition = 6
>>> s.inputs.high_pass_filter_cutoff = 128.
>>> evs_run2 = Bunch(conditions=['cond1'], onsets=[[2, 50, 100, 180]],
↳ durations=[[1]])
>>> evs_run3 = Bunch(conditions=['cond1'], onsets=[[30, 40, 100, 150]],
↳ durations=[[1]])
>>> s.inputs.subject_info = [evs_run2, evs_run3]
```

Using pmod:

```
>>> evs_run2 = Bunch(conditions=['cond1', 'cond2'], onsets=[[2, 50], [100, 180]],
↳ durations=[[0], [0]], pmod=[Bunch(name=['amp'], poly=[2], param=[[1, 2]]),
↳ None])
>>> evs_run3 = Bunch(conditions=['cond1', 'cond2'], onsets=[[20, 120], [80, 160]],
↳ durations=[[0], [0]], pmod=[Bunch(name=['amp'], poly=[2], param=[[1, 2]]),
↳ None])
>>> s.inputs.subject_info = [evs_run2, evs_run3]
```

Inputs:

```
[Mandatory]
event_files: (a list of items which are a list of items which are an
    existing file name)
    List of event description files 1, 2 or 3 column format
    corresponding to onsets, durations and amplitudes
    mutually_exclusive: subject_info, event_files
functional_runs: (a list of items which are a list of items which are
    an existing file name or an existing file name)
    Data files for model. List of 4D files or list of list of 3D files
    per session
high_pass_filter_cutoff: (a float)
    High-pass filter cutoff in secs
input_units: ('secs' or 'scans')
    Units of event onsets and durations (secs or scans). Output units
    are always in secs
subject_info: (a list of items which are a Bunch or None)
    Bunch or List(Bunch) subject-specific condition information. see
    :ref:`SpecifyModel` or SpecifyModel.__doc__ for details
    mutually_exclusive: subject_info, event_files
time_repetition: (a float)
    Time between the start of one volume to the start of the next image
    volume.

[Optional]
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
outlier_files: (a list of items which are an existing file name)
    Files containing scan outlier indices that should be tossed
parameter_source: ('SPM' or 'FSL' or 'AFNI' or 'FSFAST' or 'NIPY',
    nipyype default value: SPM)
    Source of motion parameters
```



```

realignement_parameters: (a list of items which are an existing file
                           name)
                           Realignement parameters returned by motion correction algorithm

```

Outputs:

```

session_info: (any value)
               Session info for level1designs

```

SpecifySPMModel

[Link to code](#)

Adds SPM specific options to SpecifyModel

adds:

```

concatenate_runs
output_units

```

Examples

```

>>> from nipy.algorithms import modelgen
>>> from nipy.interfaces.base import Bunch
>>> s = modelgen.SpecifySPMModel()
>>> s.inputs.input_units = 'secs'
>>> s.inputs.output_units = 'scans'
>>> s.inputs.high_pass_filter_cutoff = 128.
>>> s.inputs.functional_runs = ['functional2.nii', 'functional3.nii']
>>> s.inputs.time_repetition = 6
>>> s.inputs.concatenate_runs = True
>>> evs_run2 = Bunch(conditions=['cond1'], onsets=[[2, 50, 100, 180]],
↳durations=[[1]])
>>> evs_run3 = Bunch(conditions=['cond1'], onsets=[[30, 40, 100, 150]],
↳durations=[[1]])
>>> s.inputs.subject_info = [evs_run2, evs_run3]

```

Inputs:

```

[Mandatory]
event_files: (a list of items which are a list of items which are an
              existing file name)
              List of event description files 1, 2 or 3 column format
              corresponding to onsets, durations and amplitudes
              mutually_exclusive: subject_info, event_files
functional_runs: (a list of items which are a list of items which are
                  an existing file name or an existing file name)
                  Data files for model. List of 4D files or list of list of 3D files
                  per session
high_pass_filter_cutoff: (a float)
                          High-pass filter cutoff in secs
input_units: ('secs' or 'scans')
              Units of event onsets and durations (secs or scans). Output units
              are always in secs
subject_info: (a list of items which are a Bunch or None)
              Bunch or List(Bunch) subject-specific condition information. see
              :ref:`SpecifyModel` or SpecifyModel.__doc__ for details
              mutually_exclusive: subject_info, event_files
time_repetition: (a float)

```

Time between the start of one volume to the start of the next image volume.

[Optional]

concatenate_runs: (a boolean, nipy default value: False)

Concatenate all runs to look like a single session.

ignore_exception: (a boolean, nipy default value: False)

Print an error message instead of throwing an exception in case the interface fails to run

outlier_files: (a list of items which are an existing file name)

Files containing scan outlier indices that should be tossed

output_units: ('secs' or 'scans', nipy default value: secs)

Units of design event onsets and durations (secs or scans)

parameter_source: ('SPM' or 'FSL' or 'AFNI' or 'FSFAST' or 'NIPY', nipy default value: SPM)

Source of motion parameters

realignment_parameters: (a list of items which are an existing file name)

Realignment parameters returned by motion correction algorithm

Outputs:

session_info: (any value)

Session info for level1designs

SpecifySparseModel

[Link to code](#)

Specify a sparse model that is compatible with spm/fsl designers

References

sparse-sampling fMRI experiments. Front. Neurosci. 7:55 <http://journal.frontiersin.org/Journal/10.3389/fnins.2013.00055/abstract>

Examples

```
>>> from nipy.algorithms import modelgen
>>> from nipy.interfaces.base import Bunch
>>> s = modelgen.SpecifySparseModel()
>>> s.inputs.input_units = 'secs'
>>> s.inputs.functional_runs = ['functional2.nii', 'functional3.nii']
>>> s.inputs.time_repetition = 6
>>> s.inputs.time_acquisition = 2
>>> s.inputs.high_pass_filter_cutoff = 128.
>>> s.inputs.model_hrf = True
>>> evs_run2 = Bunch(conditions=['cond1'], onsets=[[2, 50, 100, 180]],
↳durations=[[1]])
>>> evs_run3 = Bunch(conditions=['cond1'], onsets=[[30, 40, 100, 150]],
↳durations=[[1]])
>>> s.inputs.subject_info = [evs_run2, evs_run3]
```

Inputs:

[Mandatory]

event_files: (a list of items which are a list of items which are an

```

    existing file name)
    List of event description files 1, 2 or 3 column format
    corresponding to onsets, durations and amplitudes
    mutually_exclusive: subject_info, event_files
functional_runs: (a list of items which are a list of items which are
    an existing file name or an existing file name)
    Data files for model. List of 4D files or list of list of 3D files
    per session
high_pass_filter_cutoff: (a float)
    High-pass filter cutoff in secs
input_units: ('secs' or 'scans')
    Units of event onsets and durations (secs or scans). Output units
    are always in secs
subject_info: (a list of items which are a Bunch or None)
    Bunch or List(Bunch) subject-specific condition information. see
    :ref:`SpecifyModel` or SpecifyModel.__doc__ for details
    mutually_exclusive: subject_info, event_files
time_acquisition: (a float)
    Time in seconds to acquire a single image volume
time_repetition: (a float)
    Time between the start of one volume to the start of the next image
    volume.

[Optional]
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
model_hrf: (a boolean)
    Model sparse events with hrf
outlier_files: (a list of items which are an existing file name)
    Files containing scan outlier indices that should be tossed
parameter_source: ('SPM' or 'FSL' or 'AFNI' or 'FSFAST' or 'NIPY',
    nipyype default value: SPM)
    Source of motion parameters
realignment_parameters: (a list of items which are an existing file
    name)
    Realignment parameters returned by motion correction algorithm
save_plot: (a boolean)
    Save plot of sparse design calculation (requires matplotlib)
scale_regressors: (a boolean, nipyype default value: True)
    Scale regressors by the peak
scan_onset: (a float, nipyype default value: 0.0)
    Start of scanning relative to onset of run in secs
stimuli_as_impulses: (a boolean, nipyype default value: True)
    Treat each stimulus to be impulse-like
use_temporal_deriv: (a boolean)
    Create a temporal derivative in addition to regular regressor
    requires: model_hrf
volumes_in_cluster: (a long integer >= 1, nipyype default value: 1)
    Number of scan volumes in a cluster

```

Outputs:

```

session_info: (any value)
    Session info for level1designs
sparse_png_file: (a file name)
    PNG file showing sparse design
sparse_svg_file: (a file name)

```

SVG file showing sparse design

gcd()

[Link to code](#)

Returns the greatest common divisor of two integers
uses Euclid's algorithm

```
>>> gcd(4, 5)
~
>>> gcd(4, 8)
~
>>> gcd(22, 55)
~~
```

gen_info()

[Link to code](#)

Generate subject_info structure from a list of event files

orth()

[Link to code](#)

Orthogonalize y_in with respect to x_in.

```
>>> orth_expected = np.array([1.7142857142857144, 0.42857142857142883,
↪      -0.85714285714285676])
>>> err = np.abs(np.array(orth([1, 2, 3],[4, 5, 6]) - orth_expected))
>>> all(err < np.finfo(float).eps)
True
```

scale_timings()

[Link to code](#)

Scales timings given input and output units (scans/secs)

Parameters

timelist: list of times to scale input_units: 'secs' or 'scans' output_units: Ibid. time_repetition: float in seconds

spm_hrf()

[Link to code](#)

python implementation of spm_hrf

see spm_hrf for implementation details

% RT - scan repeat time % p - parameters of the response function (two gamma % functions) % defaults
(seconds) % p(0) - delay of response (relative to onset) 6 % p(1) - delay of undershoot (relative to onset) 16 %
p(2) - dispersion of response 1 % p(3) - dispersion of undershoot 1 % p(4) - ratio of response to undershoot 6 %
p(5) - onset (seconds) 0 % p(6) - length of kernel (seconds) 32 ~ % hrf - hemodynamic response function % p -
parameters of the response function

the following code using `scipy.stats.distributions.gamma` doesn't return the same result as the `spm_Gpdf` function

```
hrf = gamma.pdf(u, p[0]/p[2], scale=dt/p[2]) -  
      gamma.pdf(u, p[1]/p[3], scale=dt/p[3])/p[4]
```

```
>>> print(spm_hrf(2))  
[ 0.00000000e+00  8.65660810e-02  3.74888236e-01  3.84923382e-01  
 2.16117316e-01  7.68695653e-02  1.62017720e-03 -3.06078117e-02  
-3.73060781e-02 -3.08373716e-02 -2.05161334e-02 -1.16441637e-02  
-5.82063147e-03 -2.61854250e-03 -1.07732374e-03 -4.10443522e-04  
-1.46257507e-04]
```


ArtifactDetect

[Link to code](#)

Detects outliers in a functional imaging series

Uses intensity and motion parameters to infer outliers. If *use_norm* is True, it computes the movement of the center of each face a cuboid centered around the head and returns the maximal movement across the centers.

Examples

```
>>> ad = ArtifactDetect()
>>> ad.inputs.realigned_files = 'functional.nii'
>>> ad.inputs.realignment_parameters = 'functional.par'
>>> ad.inputs.parameter_source = 'FSL'
>>> ad.inputs.norm_threshold = 1
>>> ad.inputs.use_differences = [True, False]
>>> ad.inputs.zintensity_threshold = 3
>>> ad.run()
```

Inputs:

```
[Mandatory]
mask_type: ('spm_global' or 'file' or 'thresh')
    Type of mask that should be used to mask the functional data.
    *spm_global* uses an spm_global like calculation to determine the
    brain mask. *file* specifies a brain mask file (should be an image
    file consisting of 0s and 1s). *thresh* specifies a threshold to
    use. By default all voxels are used, unless one of these mask types
    are defined.
norm_threshold: (a float)
    Threshold to use to detect motion-related outliers when composite
    motion is being used
    mutually_exclusive: rotation_threshold, translation_threshold
parameter_source: ('SPM' or 'FSL' or 'AFNI' or 'NiPy' or 'FSFAST')
    Source of movement parameters
realigned_files: (a list of items which are an existing file name)
    Names of realigned functional data files
```

```

realignment_parameters: (a list of items which are an existing file
    name)
    Names of realignment parameterscorresponding to the functional data
    files
rotation_threshold: (a float)
    Threshold (in radians) to use to detect rotation-related outliers
    mutually_exclusive: norm_threshold
translation_threshold: (a float)
    Threshold (in mm) to use to detect translation-related outliers
    mutually_exclusive: norm_threshold
zintensity_threshold: (a float)
    Intensity Z-threshold use to detection images that deviate from the
    mean

[Optional]
bound_by_brainmask: (a boolean, nipy default value: False)
    use the brain mask to determine bounding boxfor composite norm
    (worksfor SPM and Nipy - currentlyinaccurate for FSL, AFNI
global_threshold: (a float, nipy default value: 8.0)
    use this threshold when mask type equal's spm_global
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
intersect_mask: (a boolean)
    Intersect the masks when computed from spm_global.
mask_file: (an existing file name)
    Mask file to be used if mask_type is 'file'.
mask_threshold: (a float)
    Mask threshold to be used if mask_type is 'thresh'.
plot_type: ('png' or 'svg' or 'eps' or 'pdf', nipy default value:
    png)
    file type of the outlier plot
save_plot: (a boolean, nipy default value: True)
    save plots containing outliers
use_differences: (a list of items which are a bool or None, nipy
    default value: [True, False])
    Use differences between successive motion (first element)and
    intensity paramter (second element) estimates in orderto determine
    outliers. (default is [True, False])
use_norm: (a boolean, nipy default value: True)
    Uses a composite of the motion parameters in order to determine
    outliers.
    requires: norm_threshold

```

Outputs:

```

displacement_files: (a list of items which are a file name)
    One image file for each functional run containing the
    voxeldisplacement timeseries
intensity_files: (a list of items which are an existing file name)
    One file for each functional run containing the global intensity
    values determined from the brainmask
mask_files: (a list of items which are a file name)
    One image file for each functional run containing the maskused for
    global signal calculation
norm_files: (a list of items which are a file name)
    One file for each functional run containing the composite norm
outlier_files: (a list of items which are an existing file name)

```



```

    One file for each functional run containing a list of 0-based
    indices corresponding to outlier volumes
plot_files: (a list of items which are a file name)
    One image file for each functional run containing the detected
    outliers
statistic_files: (a list of items which are an existing file name)
    One file for each functional run containing information about the
    different types of artifacts and if design info is provided then
    details of stimulus correlated motion and a listing or artifacts by
    event type.

```

StimulusCorrelation

[Link to code](#)

Determines if stimuli are correlated with motion or intensity parameters.

Currently this class supports an SPM generated design matrix and requires intensity parameters. This implies that one must run *ArtifactDetect* and *Level1Design* prior to running this or provide an SPM.mat file and intensity parameters through some other means.

Examples

```

>>> sc = StimulusCorrelation()
>>> sc.inputs.realignment_parameters = 'functional.par'
>>> sc.inputs.intensity_values = 'functional.rms'
>>> sc.inputs.spm_mat_file = 'SPM.mat'
>>> sc.inputs.concatenated_design = False
>>> sc.run()

```

Inputs:

```

[Mandatory]
concatenated_design: (a boolean)
    state if the design matrix contains concatenated sessions
intensity_values: (a list of items which are an existing file name)
    Name of file containing intensity values
realignment_parameters: (a list of items which are an existing file
    name)
    Names of realignment parameters corresponding to the functional data
    files
spm_mat_file: (an existing file name)
    SPM mat file (use pre-estimate SPM.mat file)

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run

```

Outputs:

```

stimcorr_files: (a list of items which are an existing file name)
    List of files containing correlation values

```


interfaces.afni.preprocess

Allineate

[Link to code](#)

Wraps command **3dAllineate**

Program to align one dataset (the ‘source’) to a base dataset

For complete details, see the [3dAllineate Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> allineate = afni.Allineate()
>>> allineate.inputs.in_file = 'functional.nii'
>>> allineate.inputs.out_file = 'functional_allineate.nii'
>>> allineate.inputs.in_matrix = 'cmatrix.mat'
>>> allineate.cmdline
'3dAllineate -lDmatrix_apply cmatrix.mat -prefix functional_allineate.nii -source_
↪functional.nii'
>>> res = allineate.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dAllineate
         flag: -source %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
autobox: (a boolean)
          Expand the -automask function to enclose a rectangular box that
          holds the irregular mask.
          flag: -autobox
```

```

automask: (an integer (int or long))
    Compute a mask function, set a value for dilation or 0.
    flag: -automask+%d
autoweight: (a unicode string)
    Compute a weight function using the 3dAutomask algorithm plus some
    blurring of the base image.
    flag: -autoweight%s
center_of_mass: (a unicode string)
    Use the center-of-mass calculation to bracket the shifts.
    flag: -cmass%s
check: (a list of items which are 'leastsq' or 'ls' or 'mutualinfo'
    or 'mi' or 'corratio_mul' or 'crM' or 'norm_mutualinfo' or 'nmi' or
    'hellinger' or 'hel' or 'corratio_add' or 'crA' or 'corratio_uns'
    or 'crU')
    After cost functional optimization is done, start at the final
    parameters and RE-optimize using this new cost functions. If the
    results are too different, a warning message will be printed.
    However, the final parameters from the original optimization will be
    used to create the output dataset.
    flag: -check %s
convergence: (a float)
    Convergence test in millimeters (default 0.05mm).
    flag: -conv %f
cost: ('leastsq' or 'ls' or 'mutualinfo' or 'mi' or 'corratio_mul' or
    'crM' or 'norm_mutualinfo' or 'nmi' or 'hellinger' or 'hel' or
    'corratio_add' or 'crA' or 'corratio_uns' or 'crU')
    Defines the 'cost' function that defines the matching between the
    source and the base
    flag: -cost %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
epi: (a boolean)
    Treat the source dataset as being composed of warped EPI slices, and
    the base as comprising anatomically 'true' images. Only phase-
    encoding direction image shearing and scaling will be allowed with
    this option.
    flag: -EPI
final_interpolation: ('nearestneighbour' or 'linear' or 'cubic' or
    'quintic' or 'wsinc5')
    Defines interpolation method used to create the output dataset
    flag: -final %s
fine_blur: (a float)
    Set the blurring radius to use in the fine resolution pass to 'x'
    mm. A small amount (1-2 mm?) of blurring at the fine step may help
    with convergence, if there is some problem, especially if the base
    volume is very noisy. [Default == 0 mm = no blurring at the final
    alignment pass]
    flag: -fineblur %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_matrix: (a file name)
    matrix to align input file
    flag: -lDmatrix_apply %s, position: -3
in_param_file: (an existing file name)
    Read warp parameters from file and apply them to the source dataset,

```

```

        and produce a new dataset
        flag: -1Dparam_apply %s
interpolation: ('nearestneighbour' or 'linear' or 'cubic' or
               'quintic')
        Defines interpolation method to use during matching
        flag: -interp %s
master: (an existing file name)
        Write the output dataset on the same grid as this file.
        flag: -master %s
newgrid: (a float)
        Write the output dataset using isotropic grid spacing in mm.
        flag: -newgrid %f
nmatch: (an integer (int or long))
        Use at most n scattered points to match the datasets.
        flag: -nmatch %d
no_pad: (a boolean)
        Do not use zero-padding on the base image.
        flag: -nopad
nomask: (a boolean)
        Don't compute the autoweight/mask; if -weight is not also used, then
        every voxel will be counted equally.
        flag: -nomask
nwarp: ('bilinear' or 'cubic' or 'quintic' or 'heptic' or 'nonic' or
        'poly3' or 'poly5' or 'poly7' or 'poly9')
        Experimental nonlinear warping: bilinear or legendre poly.
        flag: -nwarp %s
nwarp_fixdep: (a list of items which are 'X' or 'Y' or 'Z' or 'I' or
               'J' or 'K')
        To fix non-linear warp dependency along directions.
        flag: -nwarp_fixdep%s
nwarp_fixmot: (a list of items which are 'X' or 'Y' or 'Z' or 'I' or
               'J' or 'K')
        To fix motion along directions.
        flag: -nwarp_fixmot%s
one_pass: (a boolean)
        Use only the refining pass -- do not try a coarse resolution pass
        first. Useful if you know that only small amounts of image alignment
        are needed.
        flag: -onepass
out_file: (a file name)
        output file from 3dAllineate
        flag: -prefix %s, position: -2
out_matrix: (a file name)
        Save the transformation matrix for each volume.
        flag: -1Dmatrix_save %s
out_param_file: (a file name)
        Save the warp parameters in ASCII (.1D) format.
        flag: -1Dparam_save %s
out_weight_file: (a file name)
        Write the weight volume to disk as a dataset
        flag: -wtprefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
reference: (an existing file name)
        file to be used as reference, the first volume will be used if not
        given the reference will be the first volume of in_file.
        flag: -base %s
replacebase: (a boolean)

```

```

        If the source has more than one volume, then after the first volume
        is aligned to the base.
        flag: -replacebase
replacemeth: ('leastsq' or 'ls' or 'mutualinfo' or 'mi' or
             'corratio_mul' or 'crM' or 'norm_mutualinfo' or 'nmi' or
             'hellinger' or 'hel' or 'corratio_add' or 'crA' or 'corratio_uns'
             or 'crU')
        After first volume is aligned, switch method for later volumes. For
        use with '-replacebase'.
        flag: -replacemeth %s
source_automask: (an integer (int or long))
        Automatically mask the source dataset with dilation or 0.
        flag: -source_automask+%d
source_mask: (an existing file name)
        mask the input dataset
        flag: -source_mask %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
two_best: (an integer (int or long))
        In the coarse pass, use the best 'bb' set of initialpoints to search
        for the starting point for the finepass. If bb==0, then no search is
        made for the beststarting point, and the identity transformation
        isused as the starting point. [Default=5; min=0 max=11]
        flag: -twobest %d
two_blur: (a float)
        Set the blurring radius for the first pass in mm.
        flag: -twoblur
two_first: (a boolean)
        Use -twopass on the first image to be registered, and then on all
        subsequent images from the source dataset, use results from the
        first image's coarse pass to start the fine pass.
        flag: -twofirst
two_pass: (a boolean)
        Use a two pass alignment strategy for all volumes, searching for a
        large rotation+shift and then refining the alignment.
        flag: -twopass
usetemp: (a boolean)
        temporary file use
        flag: -usetemp
warp_type: ('shift_only' or 'shift_rotate' or 'shift_rotate_scale' or
           'affine_general')
        Set the warp type.
        flag: -warp %s
warpfreeze: (a boolean)
        Freeze the non-rigid body parameters after first volume.
        flag: -warpfreeze
weight_file: (an existing file name)
        Set the weighting for each voxel in the base dataset; larger weights
        mean that voxel count more in the cost function. Must be defined on
        the same grid as the base dataset
        flag: -weight %s
zclip: (a boolean)
        Replace negative values in the input datasets (source & base) with
        zero.
        flag: -zclip

```

Outputs:

```
matrix: (a file name)
        matrix to align input file
out_file: (a file name)
        output image file name
```

References:: None None

AutoTcorrelate

[Link to code](#)

Wraps command **3dAutoTcorrelate**

Computes the correlation coefficient between the time series of each pair of voxels in the input dataset, and stores the output into a new anatomical bucket dataset [scaled to shorts to save memory space].

For complete details, see the [3dAutoTcorrelate Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> corr = afni.AutoTcorrelate()
>>> corr.inputs.in_file = 'functional.nii'
>>> corr.inputs.polort = -1
>>> corr.inputs.eta2 = True
>>> corr.inputs.mask = 'mask.nii'
>>> corr.inputs.mask_only_targets = True
>>> corr.cmdline
'3dAutoTcorrelate -eta2 -mask mask.nii -mask_only_targets -prefix functional_
↳similarity_matrix.1D -polort -1 functional.nii'
>>> res = corr.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        timeseries x space (volume or surface) file
        flag: %s, position: -1

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
eta2: (a boolean)
        eta^2 similarity
        flag: -eta2
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
mask: (an existing file name)
        mask of voxels
        flag: -mask %s
mask_only_targets: (a boolean)
        use mask only on targets voxels
        flag: -mask_only_targets
        mutually_exclusive: mask_source
mask_source: (an existing file name)
```

```

        mask for source voxels
        flag: -mask_source %s
        mutually_exclusive: mask_only_targets
out_file: (a file name)
        output image file name
        flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
polort: (an integer (int or long))
        Remove polynomial trend of order m or -1 for no detrending
        flag: -polort %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:: None None

Automask[Link to code](#)Wraps command **3dAutomask**

Create a brain-only mask of the image using AFNI 3dAutomask command

For complete details, see the [3dAutomask Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> automask = afni.Automask()
>>> automask.inputs.in_file = 'functional.nii'
>>> automask.inputs.dilate = 1
>>> automask.inputs.outputtype = 'NIFTI'
>>> automask.cmdline
'3dAutomask -apply_prefix functional_masked.nii -dilate 1 -prefix functional_mask.
↳nii functional.nii'
>>> res = automask.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file to 3dAutomask
        flag: %s, position: -1

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
brain_file: (a file name)
        output file from 3dAutomask
        flag: -apply_prefix %s
clfrac: (a float)
        sets the clip level fraction (must be 0.1-0.9). A small value will

```



```

        tend to make the mask larger [default = 0.5].
        flag: -clfrac %s
dilate: (an integer (int or long))
        dilate the mask outwards
        flag: -dilate %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
erode: (an integer (int or long))
        erode the mask inwards
        flag: -erode %s
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
out_file: (a file name)
        output image file name
        flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

brain_file: (an existing file name)
        brain file (skull stripped)
out_file: (an existing file name)
        mask file

```

References:: None None

Bandpass[Link to code](#)Wraps command **3dBandpass**

Program to lowpass and/or highpass each voxel time series in a dataset, offering more/different options than Fourier

For complete details, see the [3dBandpass Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> from nipy.testing import example_data
>>> bandpass = afni.Bandpass()
>>> bandpass.inputs.in_file = 'functional.nii'
>>> bandpass.inputs.highpass = 0.005
>>> bandpass.inputs.lowpass = 0.1
>>> bandpass.cmdline
'3dBandpass -prefix functional_bp 0.005000 0.100000 functional.nii'
>>> res = bandpass.run()

```

Inputs:

```

[Mandatory]
highpass: (a float)

```

```

        highpass
        flag: %f, position: -3
in_file: (an existing file name)
        input file to 3dBandpass
        flag: %s, position: -1
lowpass: (a float)
        lowpass
        flag: %f, position: -2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
automask: (a boolean)
        Create a mask from the input dataset.
        flag: -automask
blur: (a float)
        Blur (inside the mask only) with a filter width (FWHM) of 'fff'
        millimeters.
        flag: -blur %f
despike: (a boolean)
        Despike each time series before other processing. Hopefully, you
        don't actually need to do this, which is why it is optional.
        flag: -despike
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
localPV: (a float)
        Replace each vector by the local Principal Vector (AKA first
        singular vector) from a neighborhood of radius 'rrr' millimeters.
        Note that the PV time series is L2 normalized. This option is mostly
        for Bob Cox to have fun with.
        flag: -localPV %f
mask: (an existing file name)
        mask file
        flag: -mask %s, position: 2
nfft: (an integer (int or long))
        Set the FFT length [must be a legal value].
        flag: -nfft %d
no_detrend: (a boolean)
        Skip the quadratic detrending of the input that occurs before the
        FFT-based bandpassing. You would only want to do this if the dataset
        had been detrended already in some other program.
        flag: -nodetrend
normalize: (a boolean)
        Make all output time series have L2 norm = 1 (i.e., sum of squares =
        1).
        flag: -norm
notrans: (a boolean)
        Don't check for initial positive transients in the data. The test is
        a little slow, so skipping it is OK, if you KNOW the data time
        series are transient-free.
        flag: -notrans
orthogonalize_dset: (an existing file name)

```

```

        Orthogonalize each voxel to the corresponding voxel time series in
        dataset 'fset', which must have the same spatial and temporal grid
        structure as the main input dataset. At present, only one '-dsort'
        option is allowed.
        flag: -dsort %s
    orthogonalize_file: (a list of items which are an existing file name)
        Also orthogonalize input to columns in f.1D. Multiple '-ort' options
        are allowed.
        flag: -ort %s
    out_file: (a file name)
        output file from 3dBandpass
        flag: -prefix %s, position: 1
    outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
    terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
    tr: (a float)
        Set time step (TR) in sec [default=from dataset header].
        flag: -dt %f

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:: None None

BlurInMask[Link to code](#)Wraps command **3dBlurInMask**

Blurs a dataset spatially inside a mask. That's all. Experimental.

For complete details, see the [3dBlurInMask Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> bim = afni.BlurInMask()
>>> bim.inputs.in_file = 'functional.nii'
>>> bim.inputs.mask = 'mask.nii'
>>> bim.inputs.fwhm = 5.0
>>> bim.cmdline
'3dBlurInMask -input functional.nii -FWHM 5.000000 -mask mask.nii -prefix_
↳functional_blur'
>>> res = bim.run()

```

Inputs:

```

[Mandatory]
fwhm: (a float)
        fwhm kernel size
        flag: -FWHM %f
in_file: (an existing file name)
        input file to 3dSkullStrip
        flag: -input %s, position: 1

```

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
automask: (a boolean)
    Create an automask from the input dataset.
    flag: -automask
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
float_out: (a boolean)
    Save dataset as floats, no matter what the input data type is.
    flag: -float
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (a file name)
    Mask dataset, if desired. Blurring will occur only within the mask.
    Voxels NOT in the mask will be set to zero in the output.
    flag: -mask %s
multimask: (a file name)
    Multi-mask dataset -- each distinct nonzero value in dataset will be
    treated as a separate mask for blurring purposes.
    flag: -Mmask %s
options: (a unicode string)
    options
    flag: %s, position: 2
out_file: (a file name)
    output to the file
    flag: -prefix %s, position: -1
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
preserve: (a boolean)
    Normally, voxels not in the mask will be set to zero in the output.
    If you want the original values in the dataset to be preserved in
    the output, use this option.
    flag: -preserve
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:: None None

BlurToFWHM[Link to code](#)Wraps command **3dBlurToFWHM**

Blurs a ‘master’ dataset until it reaches a specified FWHM smoothness (approximately).

For complete details, see the [3dBlurToFWHM Documentation](#)

Examples

```
>>> from nipy.interfaces import afni
>>> blur = afni.preprocess.BlurToFWHM()
>>> blur.inputs.in_file = 'epi.nii'
>>> blur.inputs.fwhm = 2.5
>>> blur.cmdline
'3dBlurToFWHM -FWHM 2.500000 -input epi.nii -prefix epi_afni'
>>> res = blur.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        The dataset that will be smoothed
        flag: -input %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
automask: (a boolean)
          Create an automask from the input dataset.
          flag: -automask
blurmaster: (an existing file name)
            The dataset whose smoothness controls the process.
            flag: -blurmaster %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
fwhm: (a float)
      Blur until the 3D FWHM reaches this value (in mm)
      flag: -FWHM %f
fwhmxy: (a float)
        Blur until the 2D (x,y)-plane FWHM reaches this value (in mm)
        flag: -FWHMxy %f
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask: (an existing file name)
      Mask dataset, if desired. Voxels NOT in mask will be set to zero in
      output.
      flag: -blurmaster %s
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:: None None

ClipLevel

[Link to code](#)

Wraps command **3dClipLevel**

Estimates the value at which to clip the anatomical dataset so that background regions are set to zero. For complete details, see the [3dClipLevel Documentation](#).

Examples

```
>>> from nipy.interfaces.afni import preprocess
>>> cliplevel = preprocess.ClipLevel()
>>> cliplevel.inputs.in_file = 'anatomical.nii'
>>> cliplevel.cmdline
'3dClipLevel anatomical.nii'
>>> res = cliplevel.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input file to 3dClipLevel
        flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
doall: (a boolean)
       Apply the algorithm to each sub-brick separately.
       flag: -doall, position: 3
       mutually_exclusive: g, r, a, d
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipype default value: {})
        Environment variables
grad: (a file name)
      Also compute a 'gradual' clip level as a function of voxel position,
      and output that to a dataset.
      flag: -grad %s, position: 3
      mutually_exclusive: d, o, a, l, l
ignore_exception: (a boolean, nipype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mfrac: (a float)
       Use the number ff instead of 0.50 in the algorithm
       flag: -mfrac %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
clip_val: (a float)
          output
```

DegreeCentrality

[Link to code](#)

Wraps command 3dDegreeCentrality

Performs degree centrality on a dataset using a given maskfile via 3dDegreeCentrality

For complete details, see the [3dDegreeCentrality Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> degree = afni.DegreeCentrality()
>>> degree.inputs.in_file = 'functional.nii'
>>> degree.inputs.mask = 'mask.nii'
>>> degree.inputs.sparsity = 1 # keep the top one percent of connections
>>> degree.inputs.out_file = 'out.nii'
>>> degree.cmdline
'3dDegreeCentrality -mask mask.nii -prefix out.nii -sparsity 1.000000 functional.
↪nii'
>>> res = degree.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dDegreeCentrality
         flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
autoclip: (a boolean)
          Clip off low-intensity regions in the dataset
          flag: -autoclip
automask: (a boolean)
          Mask the dataset to target brain-only voxels
          flag: -automask
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask: (an existing file name)
      mask file to mask input data
      flag: -mask %s
oned_file: (a unicode string)
           output filepath to text dump of correlation matrix
           flag: -out1D %s
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
polort: (an integer (int or long))
        flag: -polort %d
sparsity: (a float)
          only take the top percent of connections
          flag: -sparsity %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
```

```

Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
thresh: (a float)
        threshold to exclude connections where corr <= thresh
flag: -thresh %f

```

Outputs:

```

oned_file: (a file name)
        The text output of the similarity matrix computed after thresholding
        with one-dimensional and ijk voxel indices, correlations, image
        extents, and affine matrix.
out_file: (an existing file name)
        output file

```

References:: None None

Despike[Link to code](#)Wraps command **3dDespike**

Removes 'spikes' from the 3D+time input dataset

For complete details, see the [3dDespike Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> despike = afni.Despike()
>>> despike.inputs.in_file = 'functional.nii'
>>> despike.cmdline
'3dDespike -prefix functional_despike functional.nii'
>>> res = despike.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file to 3dDespike
        flag: %s, position: -1

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
out_file: (a file name)
        output image file name
        flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')

```


Control terminal output: `stream` - displays to terminal immediately (default), `allatonce` - waits till command is finished to display output, `file` - writes output to file, `none` - output is ignored

Outputs:

out_file: (an existing file name)
output file

References:: None None

Detrend

[Link to code](#)

Wraps command **3dDetrend**

This program removes components from voxel time series using linear least squares

For complete details, see the [3dDetrend Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> detrend = afni.Detrend()
>>> detrend.inputs.in_file = 'functional.nii'
>>> detrend.inputs.args = '-polort 2'
>>> detrend.inputs.outputtype = 'AFNI'
>>> detrend.cmdline
'3dDetrend -polort 2 -prefix functional_detrend functional.nii'
>>> res = detrend.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input file to 3dDetrend
        flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:: None None

ECM

[Link to code](#)

Wraps command **3dECM**

Performs degree centrality on a dataset using a given maskfile via the 3dECM command

For complete details, see the [3dECM Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> ecm = afni.ECM()
>>> ecm.inputs.in_file = 'functional.nii'
>>> ecm.inputs.mask = 'mask.nii'
>>> ecm.inputs.sparsity = 0.1 # keep top 0.1% of connections
>>> ecm.inputs.out_file = 'out.nii'
>>> ecm.cmdline
'3dECM -mask mask.nii -prefix out.nii -sparsity 0.100000 functional.nii'
>>> res = ecm.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dECM
         flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
autoclip: (a boolean)
          Clip off low-intensity regions in the dataset
          flag: -autoclip
automask: (a boolean)
          Mask the dataset to target brain-only voxels
          flag: -automask
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
eps: (a float)
     sets the stopping criterion for the power iteration;  $|v_{old} - v_{new}| < \text{eps} * |v_{old}|$ ; default = 0.001
     flag: -eps %f
fecm: (a boolean)
      Fast centrality method; substantial speed increase but cannot
      accomodate thresholding; automatically selected if -thresh or
      -sparsity are not set
      flag: -fecm
full: (a boolean)
      Full power method; enables thresholding; automatically selected if
      -thresh or -sparsity are set
      flag: -full
```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    mask file to mask input data
    flag: -mask %s
max_iter: (an integer (int or long))
    sets the maximum number of iterations to use in the power iteration;
    default = 1000
    flag: -max_iter %d
memory: (a float)
    Limit memory consumption on system by setting the amount of GB to
    limit the algorithm to; default = 2GB
    flag: -memory %f
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
polort: (an integer (int or long))
    flag: -polort %d
scale: (a float)
    scale correlation coefficients in similarity matrix to after
    shifting, x >= 0.0; default = 1.0 for -full, 0.5 for -fecm
    flag: -scale %f
shift: (a float)
    shift correlation coefficients in similarity matrix to enforce non-
    negativity, s >= 0.0; default = 0.0 for -full, 1.0 for -fecm
    flag: -shift %f
sparsity: (a float)
    only take the top percent of connections
    flag: -sparsity %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thresh: (a float)
    threshold to exclude connections where corr <= thresh
    flag: -thresh %f

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:: None None

Fim[Link to code](#)Wraps command **3dfim+**

Program to calculate the cross-correlation of an ideal reference waveform with the measured FMRI time series for each voxel.

For complete details, see the [3dfim+ Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> fim = afni.Fim()
>>> fim.inputs.in_file = 'functional.nii'
>>> fim.inputs.ideal_file= 'seed.1D'
>>> fim.inputs.out_file = 'functional_corr.nii'
>>> fim.inputs.out = 'Correlation'
>>> fim.inputs.fim_thr = 0.0009
>>> fim.cmdline
'3dfim+ -input functional.nii -ideal_file seed.1D -fim_thr 0.000900 -out_
↳Correlation -bucket functional_corr.nii'
>>> res = fim.run()
```

Inputs:

```
[Mandatory]
ideal_file: (an existing file name)
            ideal time series file name
            flag: -ideal_file %s, position: 2
in_file: (an existing file name)
          input file to 3dfim+
          flag: -input %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
fim_thr: (a float)
          fim internal mask threshold value
          flag: -fim_thr %f, position: 3
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out: (a unicode string)
     Flag to output the specified parameter
     flag: -out %s, position: 4
out_file: (a file name)
           output image file name
           flag: -bucket %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
             AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:: None None

Fourier

[Link to code](#)

Wraps command **3dFourier**

Program to lowpass and/or highpass each voxel time series in a dataset, via the FFT

For complete details, see the [3dFourier Documentation](#).

Examples

```

>>> from nipy.interfaces import afni
>>> fourier = afni.Fourier()
>>> fourier.inputs.in_file = 'functional.nii'
>>> fourier.inputs.retrend = True
>>> fourier.inputs.highpass = 0.005
>>> fourier.inputs.lowpass = 0.1
>>> fourier.cmdline
'3dFourier -highpass 0.005000 -lowpass 0.100000 -prefix functional_fourier -
↳retrend functional.nii'
>>> res = fourier.run()

```

Inputs:

```

[Mandatory]
highpass: (a float)
    highpass
    flag: -highpass %f
in_file: (an existing file name)
    input file to 3dFourier
    flag: %s, position: -1
lowpass: (a float)
    lowpass
    flag: -lowpass %f

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
retrend: (a boolean)
    Any mean and linear trend are removed before filtering. This will
    restore the trend after filtering.
    flag: -retrend
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:: None None

Hist

[Link to code](#)

Wraps command **3dHist**

Computes average of all voxels in the input dataset which satisfy the criterion in the options list

For complete details, see the [3dHist Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> hist = afni.Hist()
>>> hist.inputs.in_file = 'functional.nii'
>>> hist.cmdline
'3dHist -input functional.nii -prefix functional_hist'
>>> res = hist.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dHist
         flag: -input %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
bin_width: (a float)
           bin width
           flag: -binwidth %f
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask: (an existing file name)
      matrix to align input file
      flag: -mask %s
max_value: (a float)
           maximum intensity value
           flag: -max %f
min_value: (a float)
           minimum intensity value
           flag: -min %f
nbin: (an integer (int or long))
      number of bins
      flag: -nbin %d
out_file: (a file name)
          Write histogram to nipl file with this prefix
          flag: -prefix %s
out_show: (a file name)
```

```

        output image file name
        flag: > %s, position: -1
showhist: (a boolean, nipy default value: False)
        write a text visual histogram
        flag: -showhist
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
        output file
out_show: (a file name)
        output visual histogram

```

LFCD[Link to code](#)**Wraps command 3dLFCD**

Performs degree centrality on a dataset using a given maskfile via the 3dLFCD command

For complete details, see the [3dLFCD Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> lfcd = afni.LFCD()
>>> lfcd.inputs.in_file = 'functional.nii'
>>> lfcd.inputs.mask = 'mask.nii'
>>> lfcd.inputs.thresh = 0.8 # keep all connections with corr >= 0.8
>>> lfcd.inputs.out_file = 'out.nii'
>>> lfcd.cmdline
'3dLFCD -mask mask.nii -prefix out.nii -thresh 0.800000 functional.nii'
>>> res = lfcd.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file to 3dLFCD
        flag: %s, position: -1

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
autoclip: (a boolean)
        Clip off low-intensity regions in the dataset
        flag: -autoclip
automask: (a boolean)
        Mask the dataset to target brain-only voxels
        flag: -automask
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables

```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    mask file to mask input data
    flag: -mask %s
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
polort: (an integer (int or long))
    flag: -polort %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thresh: (a float)
    threshold to exclude connections where corr <= thresh
    flag: -thresh %f

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:: None None

Maskave

[Link to code](#)Wraps command **3dmaskave**

Computes average of all voxels in the input dataset which satisfy the criterion in the options list

For complete details, see the [3dmaskave Documenta**tion**](#).

Examples

```

>>> from nipy.interfaces import afni
>>> maskave = afni.Maskave()
>>> maskave.inputs.in_file = 'functional.nii'
>>> maskave.inputs.mask= 'seed_mask.nii'
>>> maskave.inputs.quiet= True
>>> maskave.cmdline
'3dmaskave -mask seed_mask.nii -quiet functional.nii > functional_maskave.1D'
>>> res = maskave.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input file to 3dmaskave
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value

```



```

        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    matrix to align input file
    flag: -mask %s, position: 1
out_file: (a file name)
    output image file name
    flag: > %s, position: -1
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
quiet: (a boolean)
    matrix to align input file
    flag: -quiet, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:: None None

Means[Link to code](#)Wraps command **3dMean**

Takes the voxel-by-voxel mean of all input datasets using 3dMean

For complete details, see the [3dMean Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> means = afni.Mean()
>>> means.inputs.in_file_a = 'im1.nii'
>>> means.inputs.in_file_b = 'im2.nii'
>>> means.inputs.out_file = 'output.nii'
>>> means.cmdline
'3dMean im1.nii im2.nii -prefix output.nii'
>>> res = means.run()

```

Inputs:

```

[Mandatory]
in_file_a: (an existing file name)
    input file to 3dMean
    flag: %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s

```

```

count: (a boolean)
    compute count of non-zero voxels
    flag: -count
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file_b: (an existing file name)
    another input file to 3dMean
    flag: %s, position: 1
mask_inter: (a boolean)
    create intersection mask
    flag: -mask_inter
mask_union: (a boolean)
    create union mask
    flag: -mask_union
non_zero: (a boolean)
    use only non-zero values
    flag: -non_zero
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
scale: (a unicode string)
    scaling of output
    flag: -%sscale
sqr: (a boolean)
    mean square instead of value
    flag: -sqr
std_dev: (a boolean)
    calculate std dev
    flag: -stdev
summ: (a boolean)
    take sum, (not average)
    flag: -sum
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:: None None

OutlierCount

[Link to code](#)Wraps command **3dToutcount**

Calculates number of ‘outliers’ a 3D+time dataset, at each time point, and writes the results to stdout.

For complete details, see the [3dToutcount Documentation](#)

Examples

```
>>> from nipyype.interfaces import afni
>>> toutcount = afni.OutlierCount()
>>> toutcount.inputs.in_file = 'functional.nii'
>>> toutcount.cmdline
'3dToutcount functional.nii > functional_outliers'
>>> res = toutcount.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input dataset
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
autoclip: (a boolean, nipyype default value: False)
          clip off small voxels
          flag: -autoclip
          mutually_exclusive: in_file
automask: (a boolean, nipyype default value: False)
          clip off small voxels
          flag: -automask
          mutually_exclusive: in_file
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyype default value: {})
          Environment variables
fraction: (a boolean, nipyype default value: False)
          write out the fraction of masked voxels which are outliers at each
          timepoint
          flag: -fraction
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
interval: (a boolean, nipyype default value: False)
          write out the median + 3.5 MAD of outlier count with each timepoint
          flag: -range
legendre: (a boolean, nipyype default value: False)
          use Legendre polynomials
          flag: -legendre
mask: (an existing file name)
      only count voxels within the given mask
      flag: -mask %s
      mutually_exclusive: autoclip, automask
out_file: (a file name)
          capture standard output
          flag: > %s, position: -1
outliers_file: (a file name)
               output image file name
               flag: -save %s
polort: (an integer (int or long))
        detrend each voxel timeseries with polynomials
        flag: -polort %d
qthr: (0.0 <= a floating point number <= 1.0)
```

```
        indicate a value for q to compute alpha
        flag: -qthr %.5f
save_outliers: (a boolean, nipyne default value: False)
        enables out_file option
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
        capture standard output
        flag: > %s, position: -1
out_outliers: (an existing file name)
        output image file name
```

QualityIndex

[Link to code](#)

Wraps command **3dTqual**

Computes a ‘quality index’ for each sub-brick in a 3D+time dataset. The output is a 1D time series with the index for each sub-brick. The results are written to stdout.

For complete details, see the [3dTqual Documentation](#)

Examples

```
>>> from nipyne.interfaces import afni
>>> tqual = afni.QualityIndex()
>>> tqual.inputs.in_file = 'functional.nii'
>>> tqual.cmdline
'3dTqual functional.nii > functional_tqual'
>>> res = tqual.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input dataset
        flag: %s, position: -2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
autoclip: (a boolean, nipyne default value: False)
        clip off small voxels
        flag: -autoclip
        mutually_exclusive: mask
automask: (a boolean, nipyne default value: False)
        clip off small voxels
        flag: -automask
        mutually_exclusive: mask
clip: (a float)
        clip off values below
        flag: -clip %f
environ: (a dictionary with keys which are a bytes or None or a value
```

```

        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interval: (a boolean, nipy default value: False)
    write out the median + 3.5 MAD of outlier count with each timepoint
    flag: -range
mask: (an existing file name)
    compute correlation only across masked voxels
    flag: -mask %s
    mutually_exclusive: autoclip, automask
out_file: (a file name)
    capture standard output
    flag: > %s, position: -1
quadrant: (a boolean, nipy default value: False)
    Similar to -spearman, but using 1 minus the quadrant correlation
    coefficient as the quality index.
    flag: -quadrant
spearman: (a boolean, nipy default value: False)
    Quality index is 1 minus the Spearman (rank) correlation coefficient
    of each sub-brick with the median sub-brick. (default).
    flag: -spearman
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
    file containing the captured standard output

```

QwarpPlusMinus[Link to code](#)

Wraps command **3dQwarp -prefix Qwarp.nii.gz -plusminus**

A version of 3dQwarp for performing field susceptibility correction using two images with opposing phase encoding directions.

For complete details, see the [3dQwarp Documentation](#).

Examples

```

>>> from nipy.interfaces import afni
>>> qwarp = afni.QwarpPlusMinus()
>>> qwarp.inputs.source_file = 'sub-01_dir-LR_epi.nii.gz'
>>> qwarp.inputs.nopadWARP = True
>>> qwarp.inputs.base_file = 'sub-01_dir-RL_epi.nii.gz'
>>> qwarp.cmdline
'3dQwarp -prefix Qwarp.nii.gz -plusminus -base sub-01_dir-RL_epi.nii.gz -
↳nopadWARP -source sub-01_dir-LR_epi.nii.gz'
>>> res = warp.run()

```

Inputs:

```

[Mandatory]
base_file: (an existing file name)
    Base image (opposite phase encoding direction than source image).
    flag: -base %s
source_file: (an existing file name)
    Source image (opposite phase encoding direction than base image).
    flag: -source %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
blur: (a list of from 1 to 2 items which are a float)
    Gaussian blur the input images by (FWHM) voxels before doing the
    alignment (the output dataset will not be blurred). The default is
    2.345 (for no good reason). Optionally, you can provide 2 values,
    and then the first one is applied to the base volume, the second to
    the source volume. A negative blur radius means to use 3D median
    filtering, rather than Gaussian blurring. This type of filtering
    will better preserve edges, which can be important in alignment.
    flag: -blur %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
minpatch: (an integer (int or long))
    Set the minimum patch size for warp searching to 'mm' voxels.
    flag: -minpatch %d
nopadWARP: (a boolean)
    If for some reason you require the warp volume to match the base
    volume, then use this option to have the outputWARP dataset(s)
    truncated.
    flag: -nopadWARP
noweight: (a boolean)
    If you want a binary weight (the old default), use this option. That
    is, each voxel in the base volume automask will be weighted the same
    in the computation of the cost functional.
    flag: -noweight
pblur: (a list of from 1 to 2 items which are a float)
    The fraction of the patch size that is used for the progressive blur
    by providing a value between 0 and 0.25. If you provide TWO values,
    the first fraction is used for progressively blurring the base image
    and the second for the source image.
    flag: -pblur %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

base_warp: (an existing file name)
    Field susceptibility correction warp (in 'mm') for base image.
source_warp: (an existing file name)
    Field susceptibility correction warp (in 'mm') for source image.

```

```
warped_base: (an existing file name)
    Undistorted base file.
warped_source: (an existing file name)
    Undistorted source file.
```

ROIStats

[Link to code](#)

Wraps command **3dROIstats**

Display statistics over masked regions

For complete details, see the [3dROIstats Documenta**tion**](#).

Examples

```
>>> from nipy.interfaces import afni
>>> roistats = afni.ROIStats()
>>> roistats.inputs.in_file = 'functional.nii'
>>> roistats.inputs.mask = 'skeleton_mask.nii.gz'
>>> roistats.inputs.quiet = True
>>> roistats.cmdline
'3dROIstats -quiet -mask skeleton_mask.nii.gz functional.nii'
>>> res = roistats.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input file to 3dROIstats
    flag: %s, position: -1
terminal_output: ('allatonce', nipy default value: allatonce)
    Control terminal output: `allatonce` - waits till command is finished
    to display output

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    input mask
    flag: -mask %s, position: 3
mask_f2short: (a boolean)
    Tells the program to convert a float mask to short integers, by
    simple rounding.
    flag: -mask_f2short, position: 2
quiet: (a boolean)
    execute quietly
    flag: -quiet, position: 1
```

Outputs:

```
stats: (an existing file name)
       output tab separated values file
```

Retroicor

[Link to code](#)

Wraps command **3dretroicor**

Performs Retrospective Image Correction for physiological motion effects, using a slightly modified version of the RETROICOR algorithm

The durations of the physiological inputs are assumed to equal the duration of the dataset. Any constant sampling rate may be used, but 40 Hz seems to be acceptable. This program's cardiac peak detection algorithm is rather simplistic, so you might try using the scanner's cardiac gating output (transform it to a spike wave if necessary). This program uses slice timing information embedded in the dataset to estimate the proper cardiac/respiratory phase for each slice. It makes sense to run this program before any program that may destroy the slice timings (e.g. 3dvolreg for motion correction).

For complete details, see the [3dretroicor Documenta**tion**](#).

Examples

```
>>> from nipy.interfaces import afni
>>> ret = afni.Retroicor()
>>> ret.inputs.in_file = 'functional.nii'
>>> ret.inputs.card = 'mask.1D'
>>> ret.inputs.resp = 'resp.1D'
>>> ret.inputs.outputtype = 'NIFTI'
>>> ret.cmdline
'3dretroicor -prefix functional_retroicor.nii -resp resp.1D -card mask.1D_
↳functional.nii'
>>> res = ret.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dretroicor
         flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
card: (an existing file name)
      1D cardiac data file for cardiac correction
      flag: -card %s, position: -2
cardphase: (a file name)
            Filename for 1D cardiac phase output
            flag: -cardphase %s, position: -6
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
order: (an integer (int or long))
        The order of the correction (2 is typical)
```



```

    flag: -order %s, position: -5
out_file: (a file name)
    output image file name
    flag: -prefix %s, position: 1
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
resp: (an existing file name)
    1D respiratory waveform data for correction
    flag: -resp %s, position: -3
respphase: (a file name)
    Filename for 1D resp phase output
    flag: -respphase %s, position: -7
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (an integer (int or long))
    Threshold for detection of R-wave peaks in input (Make sure it is
    above the background noise level, Try 3/4 or 4/5 times range plus
    minimum)
    flag: -threshold %d, position: -4

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:: None None

Seg

[Link to code](#)Wraps command **3dSeg**

3dSeg segments brain volumes into tissue classes. The program allows for adding a variety of global and voxelwise priors. However for the moment, only mixing fractions and MRF are documented.

For complete details, see the [3dSeg Documentation](#).

Examples

```

>>> from nipy.interfaces.afni import preprocess
>>> seg = preprocess.Seg()
>>> seg.inputs.in_file = 'structural.nii'
>>> seg.inputs.mask = 'AUTO'
>>> seg.cmdline
'3dSeg -mask AUTO -anat structural.nii'
>>> res = seg.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    ANAT is the volume to segment
    flag: -anat %s, position: -1
mask: ('AUTO' or an existing file name)
    only non-zero voxels in mask are analyzed. mask can either be a
    dataset or the string "AUTO" which would use AFNI's automask
    function to create the mask.
    flag: -mask %s, position: -2

```

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bias_classes: (a unicode string)
    A semicolon delimited string of classes that contribute to the
    estimation of the bias field
    flag: -bias_classes %s
bias_fwhm: (a float)
    The amount of blurring used when estimating the field bias with the
    Wells method
    flag: -bias_fwhm %f
blur_meth: ('BFT' or 'BIM')
    set the blurring method for bias field estimation
    flag: -blur_meth %s
bmrf: (a float)
    Weighting factor controlling spatial homogeneity of the
    classifications
    flag: -bmrf %f
classes: (a unicode string)
    CLASS_STRING is a semicolon delimited string of class labels
    flag: -classes %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
main_N: (an integer (int or long))
    Number of iterations to perform.
    flag: -main_N %d
mixfloor: (a float)
    Set the minimum value for any class's mixing fraction
    flag: -mixfloor %f
mixfrac: (a unicode string)
    MIXFRAC sets up the volume-wide (within mask) tissue fractions while
    initializing the segmentation (see IGNORE for exception)
    flag: -mixfrac %s
prefix: (a unicode string)
    the prefix for the output folder containing all output volumes
    flag: -prefix %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    output file

```

SkullStrip[Link to code](#)**Wraps command 3dSkullStrip**

A program to extract the brain from surrounding tissue from MRI T1-weighted images. TODO Add optional

arguments.

For complete details, see the [3dSkullStrip Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> skullstrip = afni.SkullStrip()
>>> skullstrip.inputs.in_file = 'functional.nii'
>>> skullstrip.inputs.args = '-o_ply'
>>> skullstrip.cmdline
'3dSkullStrip -input functional.nii -o_ply -prefix functional_skullstrip'
>>> res = skullstrip.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dSkullStrip
         flag: -input %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:: None None

TCorr1D

[Link to code](#)

Wraps command **3dTcorr1D**

Computes the correlation coefficient between each voxel time series in the input 3D+time dataset.

For complete details, see the [3dTcorr1D Documentation](#).

```
>>> from nipy.interfaces import afni
>>> tcorr1d = afni.TCorr1D()
>>> tcorr1d.inputs.xset= 'u_rclsl_Template.nii'
>>> tcorr1d.inputs.y_1d = 'seed.1D'
```

```
>>> tcorr1D.cmdline
'3dTcorr1D -prefix u_rc1s1_Template_correlation.nii.gz u_rc1s1_Template.nii
↳seed.1D'
>>> res = tcorr1D.run()
```

Inputs:

```
[Mandatory]
xset: (an existing file name)
      3d+time dataset input
      flag: %s, position: -2
y_1d: (an existing file name)
      1D time series file input
      flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipype default value: {})
          Environment variables
ignore_exception: (a boolean, nipype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
ktaub: (a boolean)
        Correlation is the Kendall's tau_b correlation coefficient
        flag: -ktaub, position: 1
        mutually_exclusive: pearson, spearman, quadrant
out_file: (a file name)
           output filename prefix
           flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
             AFNI output filetype
pearson: (a boolean)
          Correlation is the normal Pearson correlation coefficient
          flag: -pearson, position: 1
          mutually_exclusive: spearman, quadrant, ktaub
quadrant: (a boolean)
           Correlation is the quadrant correlation coefficient
           flag: -quadrant, position: 1
           mutually_exclusive: pearson, spearman, ktaub
spearman: (a boolean)
           Correlation is the Spearman (rank) correlation coefficient
           flag: -spearman, position: 1
           mutually_exclusive: pearson, quadrant, ktaub
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
           output file containing correlations
```

References:: None None

TCorrMap

[Link to code](#)

Wraps command **3dTcorrMap**

For each voxel time series, computes the correlation between it and all other voxels, and combines this set of values into the output dataset(s) in some way.

For complete details, see the [3dTcorrMap Documenta­tion](#).

Examples

```
>>> from nipy.interfaces import afni
>>> tcm = afni.TCorrMap()
>>> tcm.inputs.in_file = 'functional.nii'
>>> tcm.inputs.mask = 'mask.nii'
>>> tcm.mean_file = 'functional_meancorr.nii'
>>> tcm.cmdline
'3dTcorrMap -input functional.nii -mask mask.nii -Mean functional_meancorr.nii'
>>> res = tcm.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        flag: -input %s

[Optional]
absolute_threshold: (a file name)
        flag: -Thresh %f %s
mutually_exclusive: absolute_threshold, var_absolute_threshold,
        var_absolute_threshold_normalize
args: (a unicode string)
      Additional parameters to the command
      flag: %s
automask: (a boolean)
          flag: -automask
average_expr: (a file name)
              flag: -Aexpr %s %s
              mutually_exclusive: average_expr, average_expr_nonzero, sum_expr
average_expr_nonzero: (a file name)
                      flag: -Cexpr %s %s
                      mutually_exclusive: average_expr, average_expr_nonzero, sum_expr
bandpass: (a tuple of the form: (a float, a float))
          flag: -bpass %f %f
blur_fwhm: (a float)
           flag: -Gblur %f
correlation_maps: (a file name)
                 flag: -CorrMap %s
correlation_maps_masked: (a file name)
                        flag: -CorrMask %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
expr: (a unicode string)
histogram: (a file name)
           flag: -Hist %d %s
histogram_bin_numbers: (an integer (int or long))
ignore_exception: (a boolean, nipy default value: False)
```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
mask: (an existing file name)
    flag: -mask %s
mean_file: (a file name)
    flag: -Mean %s
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
pmean: (a file name)
    flag: -Pmean %s
polort: (an integer (int or long))
    flag: -polort %d
qmean: (a file name)
    flag: -Qmean %s
regress_out_timeseries: (a file name)
    flag: -ort %s
seeds: (an existing file name)
    flag: -seed %s
    mutually_exclusive: s, e, e, d, s, _, w, i, d, t, h
seeds_width: (a float)
    flag: -Mseed %f
    mutually_exclusive: s, e, e, d, s
sum_expr: (a file name)
    flag: -Sexpr %s %s
    mutually_exclusive: average_expr, average_expr_nonzero, sum_expr
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thresholds: (a list of items which are an integer (int or long))
var_absolute_threshold: (a file name)
    flag: -VarThresh %f %f %f %s
    mutually_exclusive: absolute_threshold, var_absolute_threshold,
    var_absolute_threshold_normalize
var_absolute_threshold_normalize: (a file name)
    flag: -VarThreshN %f %f %f %s
    mutually_exclusive: absolute_threshold, var_absolute_threshold,
    var_absolute_threshold_normalize
zmean: (a file name)
    flag: -Zmean %s

```

Outputs:

```

absolute_threshold: (a file name)
average_expr: (a file name)
average_expr_nonzero: (a file name)
correlation_maps: (a file name)
correlation_maps_masked: (a file name)
histogram: (a file name)
mean_file: (a file name)
pmean: (a file name)
qmean: (a file name)
sum_expr: (a file name)
var_absolute_threshold: (a file name)
var_absolute_threshold_normalize: (a file name)

```

```
zmean: (a file name)
```

References:: None None

TCorrelate

[Link to code](#)

Wraps command **3dTcorrelate**

Computes the correlation coefficient between corresponding voxel time series in two input 3D+time datasets 'xset' and 'yset'

For complete details, see the [3dTcorrelate Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> tcorrelate = afni.TCorrelate()
>>> tcorrelate.inputs.xset= 'u_rcls1_Template.nii'
>>> tcorrelate.inputs.yset = 'u_rcls2_Template.nii'
>>> tcorrelate.inputs.out_file = 'functional_tcorrelate.nii.gz'
>>> tcorrelate.inputs.polort = -1
>>> tcorrelate.inputs.pearson = True
>>> tcorrelate.cmdline
'3dTcorrelate -prefix functional_tcorrelate.nii.gz -pearson -polort -1 u_rcls1_
↳Template.nii u_rcls2_Template.nii'
>>> res = tcarrelate.run()
```

Inputs:

```
[Mandatory]
xset: (an existing file name)
      input xset
      flag: %s, position: -2
yset: (an existing file name)
      input yset
      flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
           output image file name
           flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
pearson: (a boolean)
          Correlation is the normal Pearson correlation coefficient
          flag: -pearson
polort: (an integer (int or long))
         Remove polynomial trend of order m
         flag: -polort %d
```

```
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    output file
```

References:: None None

TShift

[Link to code](#)

Wraps command **3dTshift**

Shifts voxel time series from input so that separate slices are aligned to the same temporal origin.

For complete details, see the [3dTshift Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> tshift = afni.TShift()
>>> tshift.inputs.in_file = 'functional.nii'
>>> tshift.inputs.tpattern = 'alt+z'
>>> tshift.inputs.tzero = 0.0
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern alt+z -tzero 0.0 functional.nii'
>>> res = tshift.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input file to 3dTShift
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore: (an integer (int or long))
    ignore the first set of points specified
    flag: -ignore %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interp: ('Fourier' or 'linear' or 'cubic' or 'quintic' or 'heptic')
    different interpolation methods (see 3dTShift for details) default =
    Fourier
    flag: -%s
out_file: (a file name)
    output image file name
    flag: -prefix %s
```



```

outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
rlt: (a boolean)
    Before shifting, remove the mean and linear trend
    flag: -rlt
rltplus: (a boolean)
    Before shifting, remove the mean and linear trend and later put back
    the mean
    flag: -rlt+
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tpattern: (a unicode string)
    use specified slice time pattern rather than one in header
    flag: -tpattern %s
tr: (a unicode string)
    manually set the TR. You can attach suffix "s" for seconds or "ms"
    for milliseconds.
    flag: -TR %s
tslice: (an integer (int or long))
    align each slice to time offset of given slice
    flag: -slice %s
    mutually_exclusive: tzero
tzero: (a float)
    align each slice to given time offset
    flag: -tzero %s
    mutually_exclusive: tslice

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:: None None

Volreg[Link to code](#)Wraps command **3dvolreg**

Register input volumes to a base volume using AFNI 3dvolreg command

For complete details, see the [3dvolreg Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> volreg = afni.Volreg()
>>> volreg.inputs.in_file = 'functional.nii'
>>> volreg.inputs.args = '-Fourier -twopass'
>>> volreg.inputs.zpad = 4
>>> volreg.inputs.outputtype = 'NIFTI'
>>> volreg.cmdline
'3dvolreg -Fourier -twopass -1Dfile functional.1D -1Dmatrix_save functional.aff12.
↪1D -prefix functional_volreg.nii -zpad 4 -maxdisp1D functional_md.1D functional.
↪nii'
>>> res = volreg.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file to 3dvolreg
        flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
basefile: (an existing file name)
          base file for registration
          flag: -base %s, position: -6
copyorigin: (a boolean)
            copy base file origin coords to output
            flag: -twodup
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mdld_file: (a file name)
           max displacement output file
           flag: -maxdisplD %s, position: -4
oned_file: (a file name)
           1D movement parameters output file
           flag: -1Dfile %s
oned_matrix_save: (a file name)
                  Save the matrix transformation
                  flag: -1Dmatrix_save %s
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
                 (default), `allatonce` - waits till command is finished to display
                 output, `file` - writes output to file, `none` - output is ignored
timeshift: (a boolean)
           time shift to mean slice time offset
           flag: -tshift 0
verbose: (a boolean)
         more detailed description of the process
         flag: -verbose
zpad: (an integer (int or long))
      Zeropad around the edges by 'n' voxels during rotations
      flag: -zpad %d, position: -5

```

Outputs:

```

mdld_file: (an existing file name)
           max displacement info file
oned_file: (an existing file name)
           movement parameters info file
oned_matrix_save: (an existing file name)
                  matrix transformation from base to input

```

```
out_file: (an existing file name)
          registered file
```

References:: None None

Warp

[Link to code](#)

Wraps command **3dWarp**

Use 3dWarp for spatially transforming a dataset

For complete details, see the [3dWarp Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> warp = afni.Warp()
>>> warp.inputs.in_file = 'structural.nii'
>>> warp.inputs.deoblique = True
>>> warp.inputs.out_file = 'trans.nii.gz'
>>> warp.cmdline
'3dWarp -deoblique -prefix trans.nii.gz structural.nii'
>>> res = warp.run()
```

```
>>> warp_2 = afni.Warp()
>>> warp_2.inputs.in_file = 'structural.nii'
>>> warp_2.inputs.newgrid = 1.0
>>> warp_2.inputs.out_file = 'trans.nii.gz'
>>> warp_2.cmdline
'3dWarp -newgrid 1.000000 -prefix trans.nii.gz structural.nii'
>>> res = warp_2.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dWarp
         flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
deoblique: (a boolean)
           transform dataset from oblique to cardinal
           flag: -deoblique
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
gridset: (an existing file name)
         copy grid of specified dataset
         flag: -gridset %s
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
interp: ('linear' or 'cubic' or 'NN' or 'quintic')
         spatial interpolation methods [default = linear]
```

```

        flag: -%s
matparent: (an existing file name)
            apply transformation from 3dWarpDrive
            flag: -matparent %s
mni2tta: (a boolean)
          transform dataset from MNI152 to Talaraich
          flag: -mni2tta
newgrid: (a float)
          specify grid of this size (mm)
          flag: -newgrid %f
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                Control terminal output: `stream` - displays to terminal immediately
                (default), `allatonce` - waits till command is finished to display
                output, `file` - writes output to file, `none` - output is ignored
tta2mni: (a boolean)
          transform dataset from Talairach to MNI152
          flag: -tta2mni
zpad: (an integer (int or long))
       pad input dataset with N planes of zero on all sides.
       flag: -zpad %d

```

Outputs:

```

out_file: (an existing file name)
          output file

```

References:: None None

interfaces.afni.svm

SVMTest

[Link to code](#)Wraps command **3dsvm**

Temporally predictive modeling with the support vector machine SVM Test Only For complete details, see the [3dsvm Documentation](#).

Examples

```

>>> from nipy.interfaces import afni as afni
>>> svmTest = afni.SVMTest()
>>> svmTest.inputs.in_file= 'run2+orig'
>>> svmTest.inputs.model= 'run1+orig_model'
>>> svmTest.inputs.testlabels= 'run2_categories.1D'
>>> svmTest.inputs.out_file= 'pred2_model1'
>>> res = svmTest.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         A 3D or 3D+t AFNI brick dataset to be used for testing.

```

```

    flag: -testvol %s
model: (a unicode string)
    modname is the basename for the brik containing the SVM model
    flag: -model %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
classout: (a boolean)
    Flag to specify that pname files should be integer-valued,
    corresponding to class category decisions.
    flag: -classout
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
multiclass: (a boolean)
    Specifies multiclass algorithm for classification
    flag: -multiclass %s
nodetrend: (a boolean)
    Flag to specify that pname files should not be linearly detrended
    flag: -nodetrend
nopredcensord: (a boolean)
    Flag to prevent writing predicted values for censored time-points
    flag: -nopredcensord
options: (a unicode string)
    additional options for SVM-light
    flag: %s
out_file: (a file name)
    filename for .1D prediction file(s).
    flag: -predictions %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
testlabels: (an existing file name)
    *true* class category .1D labels for the test dataset. It is used to
    calculate the prediction accuracy performance
    flag: -testlabels %s

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:: None None

SVMTrain[Link to code](#)Wraps command **3dsvm**

Temporally predictive modeling with the support vector machine SVM Train Only For complete details, see the [3dsvm Documenta**tion**](#).

Examples

```
>>> from nipy.interfaces import afni as afni
>>> svmTrain = afni.SVMTrain()
>>> svmTrain.inputs.in_file = 'run1+orig'
>>> svmTrain.inputs.trainlabels = 'run1_categories.1D'
>>> svmTrain.inputs.ttype = 'regression'
>>> svmTrain.inputs.mask = 'mask.nii'
>>> svmTrain.inputs.model = 'model_run1'
>>> svmTrain.inputs.alphas = 'alphas_run1'
>>> res = svmTrain.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    A 3D+t AFNI brik dataset to be used for training.
    flag: -trainvol %s
ttype: (a unicode string)
    tname: classification or regression
    flag: -type %s

[Optional]
alphas: (a file name)
    output alphas file name
    flag: -alpha %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
censor: (an existing file name)
    .1D censor file that allows the user to ignore certain samples in
    the training data.
    flag: -censor %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
kernel: (a unicode string)
    string specifying type of kernel function: linear, polynomial, rbf,
    sigmoid
    flag: -kernel %s
mask: (an existing file name)
    byte-format brik file used to mask voxels in the analysis
    flag: -mask %s, position: -1
max_iterations: (an integer (int or long))
    Specify the maximum number of iterations for the optimization.
    flag: -max_iterations %d
model: (a file name)
    basename for the brik containing the SVM model
    flag: -model %s
nomodelmask: (a boolean)
    Flag to enable the omission of a mask file
    flag: -nomodelmask
options: (a unicode string)
    additional options for SVM-light
    flag: %s
```

```

out_file: (a file name)
    output sum of weighted linear support vectors file name
    flag: -bucket %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
trainlabels: (an existing file name)
    .1D labels corresponding to the stimulus paradigm for the training
    data.
    flag: -trainlabels %s
w_out: (a boolean)
    output sum of weighted linear support vectors
    flag: -wout

```

Outputs:

```

alphas: (a file name)
    output alphas file name
model: (a file name)
    brik containing the SVM model file name
out_file: (a file name)
    sum of weighted linear support vectors file name

```

References:: None None

interfaces.afni.utils

AFNItO~~NIFTI~~

[Link to code](#)Wraps command **3dAFNItO~~NIFTI~~**

Converts AFNI format files to NIFTI format. This can also convert 2D or 1D data, which you can `numpy.squeeze()` to remove extra dimensions.

For complete details, see the [3dAFNItO~~NIFTI~~ Documentation](#).

Examples

```

>>> from nipy.interfaces import afni
>>> a2n = afni.AFNItONIFTI()
>>> a2n.inputs.in_file = 'afni_output.3D'
>>> a2n.inputs.out_file = 'afni_output.nii'
>>> a2n.cmdline
'3dAFNItONIFTI -prefix afni_output.nii afni_output.3D'
>>> res = a2n.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input file to 3dAFNItONIFTI
    flag: %s, position: -1

[Optional]
args: (a unicode string)

```

```
Additional parameters to the command
flag: %s
denote: (a boolean)
    When writing the AFNI extension field, remove text notes that might
    contain subject identifying information.
    flag: -denote
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
newid: (a boolean)
    Give the new dataset a new AFNI ID code, to distinguish it from the
    input dataset.
    flag: -newid
    mutually_exclusive: oldid
oldid: (a boolean)
    Give the new dataset the input datasets AFNI ID code.
    flag: -oldid
    mutually_exclusive: newid
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
pure: (a boolean)
    Do NOT write an AFNI extension field into the output file. Only use
    this option if needed. You can also use the 'nifti_tool' program to
    strip extensions from a file.
    flag: -pure
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    output file
```

References:: None None

Autobox

[Link to code](#)

Wraps command **3dAutobox**

Computes size of a box that fits around the volume. Also can be used to crop the volume to that box.

For complete details, see the [3dAutobox Documentation](#).

Examples

```
>>> from nipyne.interfaces import afni
>>> abox = afni.Autobox()
>>> abox.inputs.in_file = 'structural.nii'
>>> abox.inputs.padding = 5
>>> abox.cmdline
```



```
'3dAutobox -input structural.nii -prefix structural_autobox -npad 5'
>>> res = abox.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input file
        flag: -input %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
no_clustering: (a boolean)
               Don't do any clustering to find box. Any non-zero voxel will be
               preserved in the cropped volume. The default method uses some
               clustering to find the cropping box, and will clip off small
               isolated blobs.
               flag: -noclust
out_file: (a file name)
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
padding: (an integer (int or long))
         Number of extra voxels to pad on each side of box
         flag: -npad %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
                 (default), `allatonce` - waits till command is finished to display
                 output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          output file
x_max: (an integer (int or long))
x_min: (an integer (int or long))
y_max: (an integer (int or long))
y_min: (an integer (int or long))
z_max: (an integer (int or long))
z_min: (an integer (int or long))
```

References:: None None

BrickStat[Link to code](#)Wraps command **3dBrickStat**

Computes maximum and/or minimum voxel values of an input dataset. TODO Add optional arguments.
For complete details, see the [3dBrickStat Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> brickstat = afni.BrickStat()
>>> brickstat.inputs.in_file = 'functional.nii'
>>> brickstat.inputs.mask = 'skeleton_mask.nii.gz'
>>> brickstat.inputs.min = True
>>> brickstat.cmdline
'3dBrickStat -min -mask skeleton_mask.nii.gz functional.nii'
>>> res = brickstat.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dmaskave
         flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask: (an existing file name)
      -mask dset = use dset as mask to include/exclude voxels
      flag: -mask %s, position: 2
min: (a boolean)
     print the minimum value in dataset
     flag: -min, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
min_val: (a float)
         output
```

Calc

[Link to code](#)

Wraps command **3dcalc**

This program does voxel-by-voxel arithmetic on 3D datasets.

For complete details, see the [3dcalc Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> calc = afni.Calc()
>>> calc.inputs.in_file_a = 'functional.nii'
```

```

>>> calc.inputs.in_file_b = 'functional2.nii'
>>> calc.inputs.expr='a*b'
>>> calc.inputs.out_file = 'functional_calc.nii.gz'
>>> calc.inputs.outputtype = 'NIFTI'
>>> calc.cmdline
'3dcalc -a functional.nii -b functional2.nii -expr "a*b" -prefix functional_calc.
↪nii.gz'
>>> res = calc.run()

```

Inputs:

```

[Mandatory]
expr: (a unicode string)
    expr
    flag: -expr "%s", position: 3
in_file_a: (an existing file name)
    input file to 3dcalc
    flag: -a %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file_b: (an existing file name)
    operand file to 3dcalc
    flag: -b %s, position: 1
in_file_c: (an existing file name)
    operand file to 3dcalc
    flag: -c %s, position: 2
other: (a file name)
    other options
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
single_idx: (an integer (int or long))
    volume index for in_file_a
start_idx: (an integer (int or long))
    start index for in_file_a
    requires: stop_idx
stop_idx: (an integer (int or long))
    stop index for in_file_a
    requires: start_idx
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:: None None

Copy

[Link to code](#)

Wraps command **3dcopy**

Copies an image of one type to an image of the same or different type using 3dcopy command

For complete details, see the [3dcopy Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> copy3d = afni.Copy()
>>> copy3d.inputs.in_file = 'functional.nii'
>>> copy3d.cmdline
'3dcopy functional.nii functional_copy'
>>> res = copy3d.run()
```

```
>>> from copy import deepcopy
>>> copy3d_2 = deepcopy(copy3d)
>>> copy3d_2.inputs.outputtype = 'NIFTI'
>>> copy3d_2.cmdline
'3dcopy functional.nii functional_copy.nii'
>>> res = copy3d_2.run()
```

```
>>> copy3d_3 = deepcopy(copy3d)
>>> copy3d_3.inputs.outputtype = 'NIFTI_GZ'
>>> copy3d_3.cmdline
'3dcopy functional.nii functional_copy.nii.gz'
>>> res = copy3d_3.run()
```

```
>>> copy3d_4 = deepcopy(copy3d)
>>> copy3d_4.inputs.out_file = 'new_func.nii'
>>> copy3d_4.cmdline
'3dcopy functional.nii new_func.nii'
>>> res = copy3d_4.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dcopy
         flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
out_file: (a file name)
        output image file name
        flag: %s, position: -1
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:: None None

Eval[Link to code](#)Wraps command **1deval**

Evaluates an expression that may include columns of data from one or more text files.

For complete details, see the [1deval Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> eval = afni.Eval()
>>> eval.inputs.in_file_a = 'seed.1D'
>>> eval.inputs.in_file_b = 'resp.1D'
>>> eval.inputs.expr = 'a*b'
>>> eval.inputs.out1D = True
>>> eval.inputs.out_file = 'data_calc.1D'
>>> eval.cmdline
'1deval -a seed.1D -b resp.1D -expr "a*b" -1D -prefix data_calc.1D'
>>> res = eval.run()

```

Inputs:

```

[Mandatory]
expr: (a unicode string)
        expr
        flag: -expr "%s", position: 3
in_file_a: (an existing file name)
        input file to 1deval
        flag: -a %s, position: 0

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)

```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
in_file_b: (an existing file name)
    operand file to ldeval
    flag: -b %s, position: 1
in_file_c: (an existing file name)
    operand file to ldeval
    flag: -c %s, position: 2
other: (a file name)
    other options
out1D: (a boolean)
    output in 1D
    flag: -1D
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
single_idx: (an integer (int or long))
    volume index for in_file_a
start_idx: (an integer (int or long))
    start index for in_file_a
    requires: stop_idx
stop_idx: (an integer (int or long))
    stop index for in_file_a
    requires: start_idx
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:: None None

FWHMx[Link to code](#)Wraps command **3dFWHMx**

Unlike the older 3dFWHM, this program computes FWHMs for all sub-bricks in the input dataset, each one separately. The output for each one is written to the file specified by ‘-out’. The mean (arithmetic or geometric) of all the FWHMs along each axis is written to stdout. (A non-positive output value indicates something bad happened; e.g., FWHM in z is meaningless for a 2D dataset; the estimation method computed incoherent intermediate results.)

For complete details, see the [3dFWHMx Documentation](#).

Examples

```

>>> from nipy.interfaces import afni
>>> fwhm = afni.FWHMx()
>>> fwhm.inputs.in_file = 'functional.nii'
>>> fwhm.cmdline
'3dFWHMx -input functional.nii -out functional_subbricks.out > functional_fwhmx.
↳out '

```

```
>>> res = fwhm.run()
```

(Classic) METHOD:

- Calculate ratio of variance of first differences to data variance.
- Should be the same as 3dFWHM for a 1-brick dataset. (But the output format is simpler to use in a script.)

Note: IMPORTANT NOTE [AFNI > 16]

A completely new method for estimating and using noise smoothness values is now available in 3dFWHMx and 3dClustSim. This method is implemented in the ‘-acf’ options to both programs. ‘ACF’ stands for (spatial) AutoCorrelation Function, and it is estimated by calculating moments of differences out to a larger radius than before.

Notably, real FMRI data does not actually have a Gaussian-shaped ACF, so the estimated ACF is then fit (in 3dFWHMx) to a mixed model (Gaussian plus mono-exponential) of the form

$$ACF(r) = a * \exp(-r * r / (2 * b * b)) + (1 - a) * \exp(-r/c)$$

where r is the radius, and a, b, c are the fitted parameters. The apparent FWHM from this model is usually somewhat larger in real data than the FWHM estimated from just the nearest-neighbor differences used in the ‘classic’ analysis.

The longer tails provided by the mono-exponential are also significant. 3dClustSim has also been modified to use the ACF model given above to generate noise random fields.

Note: TL;DR or summary

The take-awaymessage is that the ‘classic’ 3dFWHMx and 3dClustSim analysis, using a pure Gaussian ACF, is not very correct for FMRI data – I cannot speak for PET or MEG data.

Warning: Do NOT use 3dFWHMx on the statistical results (e.g., ‘-bucket’) from 3dDeconvolve or 3dREMLfit!!! The function of 3dFWHMx is to estimate the smoothness of the time series NOISE, not of the statistics. This proscription is especially true if you plan to use 3dClustSim next!!

Note: Recommendations

- For FMRI statistical purposes, you DO NOT want the FWHM to reflect the spatial structure of the underlying anatomy. Rather, you want the FWHM to reflect the spatial structure of the noise. This means that the input dataset should not have anatomical (spatial) structure.
- One good form of input is the output of ‘3dDeconvolve -errts’, which is the dataset of residuals left over after the GLM fitted signal model is subtracted out from each voxel’s time series.
- If you don’t want to go to that much trouble, use ‘-detrend’ to approximately subtract out the anatomical spatial structure, OR use the output of 3dDetrend for the same purpose.
- If you do not use ‘-detrend’, the program attempts to find non-zero spatial structure in the input, and will print a warning message if it is detected.

Note: Notes on -demend

- I recommend this option, and it is not the default only for historical compatibility reasons. It may become the default someday.
- It is already the default in program 3dBlurToFWHM. This is the same detrending as done in 3dDespike; using $2*q+3$ basis functions for $q > 0$.
- If you don’t use ‘-detrend’, the program now [Aug 2010] checks if a large number of voxels are have significant nonzero means. If so, the program will print a warning message suggesting the use of ‘-detrend’, since inherent spatial structure in the image will bias the estimation of the FWHM of the image time series NOISE (which is

usually the point of using 3dFWHMx).

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input dataset
        flag: -input %s

[Optional]
acf: (a boolean or a file name or a tuple of the form: (an existing
      file name, a float), nipy default value: False)
      computes the spatial autocorrelation
      flag: -acf
args: (a unicode string)
      Additional parameters to the command
      flag: %s
arith: (a boolean)
        if in_file has more than one sub-brick, compute the final estimate
        as the arithmetic mean of the individual sub-brick FWHM estimates
        flag: -arith
        mutually_exclusive: geom
automask: (a boolean, nipy default value: False)
          compute a mask from THIS dataset, a la 3dAutomask
          flag: -automask
combine: (a boolean)
          combine the final measurements along each axis
          flag: -combine
compat: (a boolean)
         be compatible with the older 3dFWHM
         flag: -compat
demed: (a boolean)
        If the input dataset has more than one sub-brick (e.g., has a time
        axis), then subtract the median of each voxel's time series before
        processing FWHM. This will tend to remove intrinsic spatial
        structure and leave behind the noise.
        flag: -demed
        mutually_exclusive: detrend
detrend: (a boolean or an integer (int or long), nipy default
          value: False)
          instead of demed (0th order detrending), detrend to the specified
          order. If order is not given, the program picks q=NT/30. -detrend
          disables -demed, and includes -unif.
          flag: -detrend
          mutually_exclusive: demed
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
geom: (a boolean)
       if in_file has more than one sub-brick, compute the final estimate
       as the geometric mean of the individual sub-brick FWHM estimates
       flag: -geom
       mutually_exclusive: arith
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask: (an existing file name)
```



```

        use only voxels that are nonzero in mask
        flag: -mask %s
out_detrend: (a file name)
        Save the detrended file into a dataset
        flag: -detprefix %s
out_file: (a file name)
        output file
        flag: > %s, position: -1
out_subbricks: (a file name)
        output file listing the subbricks FWHM
        flag: -out %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
unif: (a boolean)
        If the input dataset has more than one sub-brick, then normalize
        each voxel's time series to have the same MAD before processing
        FWHM.
        flag: -unif

```

Outputs:

```

acf_param: (a tuple of the form: (a float, a float, a float) or a
        tuple of the form: (a float, a float, a float, a float))
        fitted ACF model parameters
fwhm: (a tuple of the form: (a float, a float, a float) or a tuple of
        the form: (a float, a float, a float, a float))
        FWHM along each axis
out_acf: (an existing file name)
        output acf file
out_detrend: (a file name)
        output file, detrended
out_file: (an existing file name)
        output file
out_subbricks: (an existing file name)
        output file (subbricks)

```

References:: None

MaskTool[Link to code](#)Wraps command **3dmask_tool**

3dmask_tool - for combining/dilating/eroding/filling masks

For complete details, see the [3dmask_tool Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> masktool = afni.MaskTool()
>>> masktool.inputs.in_file = 'functional.nii'
>>> masktool.inputs.outputtype = 'NIFTI'
>>> masktool.cmdline
'3dmask_tool -prefix functional_mask.nii -input functional.nii'
>>> res = automask.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file or files to 3dmask_tool
        flag: -input %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
count: (a boolean)
       Instead of created a binary 0/1 mask dataset, create one with counts
       of voxel overlap, i.e., each voxel will contain the number of masks
       that it is set in.
       flag: -count, position: 2
datum: ('byte' or 'short' or 'float')
       specify data type for output. Valid types are 'byte', 'short' and
       'float'.
       flag: -datum %s
dilate_inputs: (a unicode string)
              Use this option to dilate and/or erode datasets as they are read.
              ex. '5 -5' to dilate and erode 5 times
              flag: -dilate_inputs %s
dilate_results: (a unicode string)
               dilate and/or erode combined mask at the given levels.
               flag: -dilate_results %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
fill_dirs: (a unicode string)
          fill holes only in the given directions. This option is for use with
          -fill holes. should be a single string that specifies 1-3 of the
          axes using {x,y,z} labels (i.e. dataset axis order), or using the
          labels in {R,L,A,P,I,S}.
          flag: -fill_dirs %s
          requires: fill_holes
fill_holes: (a boolean)
           This option can be used to fill holes in the resulting mask, i.e.
           after all other processing has been done.
           flag: -fill_holes
frac: (a float)
      When combining masks (across datasets and sub-bricks), use this
      option to restrict the result to a certain fraction of the set of
      volumes
      flag: -frac %s
ignore_exception: (a boolean, nipy default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
inter: (a boolean)
       intersection, this means -frac 1.0
       flag: -inter
out_file: (a file name)
         output image file name
         flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
           AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                Control terminal output: `stream` - displays to terminal immediately

```

```
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
union: (a boolean)
      union, this means -frac 0
flag: -union
```

Outputs:

```
out_file: (an existing file name)
      mask file
```

References:: None None

Merge

[Link to code](#)

Wraps command **3dmerge**

Merge or edit volumes using AFNI 3dmerge command

For complete details, see the [3dmerge Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> merge = afni.Merge()
>>> merge.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> merge.inputs.blurfwhm = 4
>>> merge.inputs.doall = True
>>> merge.inputs.out_file = 'e7.nii'
>>> merge.cmdline
'3dmerge -lblur_fwhm 4 -doall -prefix e7.nii functional.nii functional2.nii'
>>> res = merge.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
      flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
blurfwhm: (an integer (int or long))
      FWHM blur value (mm)
      flag: -lblur_fwhm %d
doall: (a boolean)
      apply options to all sub-bricks in dataset
      flag: -doall
environ: (a dictionary with keys which are a bytes or None or a value
      of class 'str' and with values which are a bytes or None or a value
      of class 'str', nipy default value: {})
      Environment variables
ignore_exception: (a boolean, nipy default value: False)
      Print an error message instead of throwing an exception in case the
      interface fails to run
out_file: (a file name)
      output image file name
      flag: -prefix %s
```

```
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    output file
```

References:: None None

Notes

[Link to code](#)

Wraps command **3dNotes**

A program to add, delete, and show notes for AFNI datasets.

For complete details, see the [3dNotes Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> notes = afni.Notes()
>>> notes.inputs.in_file = 'functional.HEAD'
>>> notes.inputs.add = 'This note is added.'
>>> notes.inputs.add_history = 'This note is added to history.'
>>> notes.cmdline
'3dNotes -a "This note is added." -h "This note is added to history." functional.
↪HEAD'
>>> res = notes.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input file to 3dNotes
    flag: %s, position: -1

[Optional]
add: (a unicode string)
    note to add
    flag: -a "%s"
add_history: (a unicode string)
    note to add to history
    flag: -h "%s"
    mutually_exclusive: rep_history
args: (a unicode string)
    Additional parameters to the command
    flag: %s
delete: (an integer (int or long))
    delete note number num
    flag: -d %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    output image file name
    flag: %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
rep_history: (a unicode string)
    note with which to replace history
    flag: -HH "%s"
    mutually_exclusive: add_history
ses: (a boolean)
    print to stdout the expanded notes
    flag: -ses
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    output file

```

Refit[Link to code](#)**Wraps command 3drefit**

Changes some of the information inside a 3D dataset's header

For complete details, see the [3drefit Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> refit = afni.Refit()
>>> refit.inputs.in_file = 'structural.nii'
>>> refit.inputs.deoblique = True
>>> refit.cmdline
'3drefit -deoblique structural.nii'
>>> res = refit.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input file to 3drefit
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
deoblique: (a boolean)
    replace current transformation matrix with cardinal matrix
    flag: -deoblique
environ: (a dictionary with keys which are a bytes or None or a value

```

```

        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
space: ('TLRC' or 'MNI' or 'ORIG')
    Associates the dataset with a specific template type, e.g. TLRC,
    MNI, ORIG
    flag: -space %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xdel: (a float)
    new x voxel dimension in mm
    flag: -xdel %f
xorigin: (a unicode string)
    x distance for edge voxel offset
    flag: -xorigin %s
ydel: (a float)
    new y voxel dimension in mm
    flag: -ydel %f
yorigin: (a unicode string)
    y distance for edge voxel offset
    flag: -yorigin %s
zdel: (a float)
    new z voxel dimension in mm
    flag: -zdel %f
zorigin: (a unicode string)
    z distance for edge voxel offset
    flag: -zorigin %s

```

Outputs:

```

out_file: (an existing file name)
    output file

```

Resample[Link to code](#)**Wraps command 3dresample**

Resample or reorient an image using AFNI 3dresample command

For complete details, see the [3dresample Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> resample = afni.Resample()
>>> resample.inputs.in_file = 'functional.nii'
>>> resample.inputs.orientation= 'RPI'
>>> resample.inputs.outputtype = 'NIFTI'
>>> resample.cmdline
'3dresample -orient RPI -prefix functional_resample.nii -inset functional.nii'
>>> res = resample.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file to 3dresample
        flag: -inset %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyre default value: {})
          Environment variables
ignore_exception: (a boolean, nipyre default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
master: (a file name)
        align dataset grid to a reference file
        flag: -master %s
orientation: (a unicode string)
             new orientation code
             flag: -orient %s
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
resample_mode: ('NN' or 'Li' or 'Cu' or 'Bk')
               resampling method from set {"NN", "Li", "Cu", "Bk"}. These are for
               "Nearest Neighbor", "Linear", "Cubic" and "Blocky" interpolation,
               respectively. Default is NN.
               flag: -rmode %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
                 (default), `allatonce` - waits till command is finished to display
                 output, `file` - writes output to file, `none` - output is ignored
voxel_size: (a tuple of the form: (a float, a float, a float))
            resample to new dx, dy and dz
            flag: -dxyz %f %f %f

```

Outputs:

```

out_file: (an existing file name)
          output file

```

References:: None None

TCat[Link to code](#)Wraps command **3dTcat**

Concatenate sub-bricks from input datasets into one big 3D+time dataset.

TODO Replace InputMultiPath in_files with Traits.List, if possible. Current version adds extra whitespace.

For complete details, see the [3dTcat Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> tcat = afni.TCat()
>>> tcat.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> tcat.inputs.out_file= 'functional_tcat.nii'
>>> tcat.inputs.rlt = '+'
>>> tcat.cmdline
'3dTcat -rlt+ -prefix functional_tcat.nii functional.nii functional2.nii'
>>> res = tcat.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
    input file to 3dTcat
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
rlt: ('' or '+' or '++')
    Remove linear trends in each voxel time series loaded from each
    input dataset, SEPARATELY. Option -rlt removes the least squares fit
    of 'a+b*t' to each voxel time series. Option -rlt+ adds dataset mean
    back in. Option -rlt++ adds overall mean of all dataset timeseries
    back in.
    flag: -rlt%s, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    output file
```

References:: None None

TStat

[Link to code](#)

Wraps command **3dTstat**

Compute voxel-wise statistics using AFNI 3dTstat command

For complete details, see the [3dTstat Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> tstat = afni.TStat()
>>> tstat.inputs.in_file = 'functional.nii'
>>> tstat.inputs.args = '-mean'
>>> tstat.inputs.out_file = 'stats'
>>> tstat.cmdline
'3dTstat -mean -prefix stats functional.nii'
>>> res = tstat.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input file to 3dTstat
        flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask: (an existing file name)
      mask file
      flag: -mask %s
options: (a unicode string)
         selected statistical output
         flag: %s
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
                 (default), `allatonce` - waits till command is finished to display
                 output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:: None None

To3D

[Link to code](#)

Wraps command **to3d**

Create a 3D dataset from 2D image files using AFNI to3d command

For complete details, see the [to3d Documentation](#)

Examples

```
>>> from nipy.interfaces import afni
>>> to3d = afni.To3D()
>>> to3d.inputs.datatype = 'float'
>>> to3d.inputs.in_folder = '.'
>>> to3d.inputs.out_file = 'dicomdir.nii'
>>> to3d.inputs.filetype = 'anat'
>>> to3d.cmdline
'to3d -datum float -anat -prefix dicomdir.nii ./*.dcm'
>>> res = to3d.run()
```

Inputs:

```
[Mandatory]
in_folder: (an existing directory name)
            folder with DICOM images to convert
            flag: %s/*.dcm, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
assumemosaic: (a boolean)
              assume that Siemens image is mosaic
              flag: -assume_dicom_mosaic
datatype: ('short' or 'float' or 'byte' or 'complex')
           set output file datatype
           flag: -datum %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
filetype: ('spgr' or 'fse' or 'epan' or 'anat' or 'ct' or 'spct' or
           'pet' or 'mra' or 'bmap' or 'diff' or 'omri' or 'abuc' or 'fim' or
           'fith' or 'fico' or 'fitt' or 'fift' or 'fizt' or 'fict' or 'fibt'
           or 'fibn' or 'figt' or 'fipt' or 'fbuc')
           type of datafile being converted
           flag: -%s
funcparams: (a unicode string)
            parameters for functional data
            flag: -time:zt %s alt+z2
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
skipoutliers: (a boolean)
              skip the outliers check
              flag: -skip_outliers
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:: None None

Unifize

[Link to code](#)

Wraps command **3dUnifize**

3dUnifize - for uniformizing image intensity

- The input dataset is supposed to be a T1-weighted volume, possibly already skull-stripped (e.g., via 3dSkullStrip). However, this program can be a useful step to take BEFORE 3dSkullStrip, since the latter program can fail if the input volume is strongly shaded – 3dUnifize will (mostly) remove such shading artifacts.
- The output dataset has the white matter (WM) intensity approximately uniformized across space, and scaled to peak at about 1000.
- The output dataset is always stored in float format!
- If the input dataset has more than 1 sub-brick, only sub-brick #0 will be processed!
- Want to correct EPI datasets for nonuniformity? You can try the new and experimental [Mar 2017] ‘-EPI’ option.
- The principal motive for this program is for use in an image registration script, and it may or may not be useful otherwise.
- This program replaces the older (and very different) 3dUniformize, which is no longer maintained and may sublimiate at any moment. (In other words, we do not recommend the use of 3dUniformize.)

For complete details, see the [3dUnifize Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> unifize = afni.Unifize()
>>> unifize.inputs.in_file = 'structural.nii'
>>> unifize.inputs.out_file = 'structural_unifized.nii'
>>> unifize.cmdline
'3dUnifize -prefix structural_unifized.nii -input structural.nii'
>>> res = unifize.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dUnifize
         flag: -input %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
epi: (a boolean)
     Assume the input dataset is a T2 (or T2*) weighted EPI time series.
     After computing the scaling, apply it to ALL volumes (TRs) in the
     input dataset. That is, a given voxel will be scaled by the same
     factor at each TR. This option also implies '-noduplo' and
     '-T2'. This option turns off '-GM' if you turned it on.
     flag: -EPI
     mutually_exclusive: gm
```

```

    requires: no_duplo, t2
gm: (a boolean)
    Also scale to unifize 'gray matter' = lower intensity voxels (to aid
    in registering images from different scanners).
    flag: -GM
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
no_duplo: (a boolean)
    Do NOT use the 'duplo down' step; this can be useful for lower
    resolution datasets.
    flag: -noduplo
out_file: (a file name)
    output image file name
    flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
scale_file: (a file name)
    output file name to save the scale factor used at each voxel
    flag: -ssave %s
t2: (a boolean)
    Treat the input as if it were T2-weighted, rather than T1-weighted.
    This processing is done simply by inverting the image contrast,
    processing it as if that result were T1-weighted, and then re-
    inverting the results counts of voxel overlap, i.e., each voxel will
    contain the number of masks that it is set in.
    flag: -T2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
urad: (a float)
    Sets the radius (in voxels) of the ball used for the sneaky trick.
    Default value is 18.3, and should be changed proportionally if the
    dataset voxel size differs significantly from 1 mm.
    flag: -Urad %s

```

Outputs:

```

out_file: (an existing file name)
    unifized file
scale_file: (a file name)
    scale factor file

```

References:: None None

ZCutUp[Link to code](#)Wraps command **3dZcutup**

Cut z-slices from a volume using AFNI 3dZcutup command

For complete details, see the [3dZcutup Documenta**tion**](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> zcutup = afni.ZCutUp()
>>> zcutup.inputs.in_file = 'functional.nii'

```

```
>>> zcutup.inputs.out_file = 'functional_zcutup.nii'
>>> zcutup.inputs.keep= '0 10'
>>> zcutup.cmdline
'3dZcutup -keep 0 10 -prefix functional_zcutup.nii functional.nii'
>>> res = zcutup.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input file to 3dZcutup
        flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
keep: (a unicode string)
      slice range to keep in output
      flag: -keep %s
out_file: (a file name)
          output image file name
          flag: -prefix %s
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:: None None

interfaces.ants.legacy

GenWarpFields

[Link to code](#)

Wraps command **antsIntroduction.sh**

Inputs:

```
[Mandatory]
input_image: (an existing file name)
    input image to warp to template
    flag: -i %s
reference_image: (an existing file name)
    template file to warp to
    flag: -r %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bias_field_correction: (a boolean)
    Applies bias field correction to moving image
    flag: -n 1
dimension: (3 or 2, nipype default value: 3)
    image dimension (2 or 3)
    flag: -d %d, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
force_proceed: (a boolean)
    force script to proceed even if headers may be incompatible
    flag: -f 1
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inverse_warp_template_labels: (a boolean)
```

```

    Applies inverse warp to the template labels to estimate label
    positions in target space (use for template-based segmentation)
    flag: -l
max_iterations: (a list of items which are an integer (int or long))
    maximum number of iterations (must be list of integers in the form
    [J,K,L...]: J = coarsest resolution iterations, K = middle
    resolution iterations, L = fine resolution iterations
    flag: -m %s
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_prefix: (a unicode string, nipy default value: ants_)
    Prefix that is prepended to all output files (default = ants_)
    flag: -o %s
quality_check: (a boolean)
    Perform a quality check of the result
    flag: -q 1
similarity_metric: ('PR' or 'CC' or 'MI' or 'MSQ')
    Type of similarity metric used for registration (CC = cross
    correlation, MI = mutual information, PR = probability mapping, MSQ
    = mean square difference)
    flag: -s %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformation_model: ('GR' or 'EL' or 'SY' or 'S2' or 'EX' or 'DD'
    or 'RI' or 'RA', nipy default value: GR)
    Type of transformation model used for registration (EL = elastic
    transformation model, SY = SyN with time, arbitrary number of time
    points, S2 = SyN with time optimized for 2 time points, GR = greedy
    SyN, EX = exponential, DD = diffeomorphic demons style exponential
    mapping, RI = purely rigid, RA = affine rigid
    flag: -t %s

```

Outputs:

```

affine_transformation: (an existing file name)
    affine (prefix_Affine.txt)
input_file: (an existing file name)
    input image (prefix_repaired.nii)
inverse_warp_field: (an existing file name)
    inverse warp field (prefix_InverseWarp.nii)
output_file: (an existing file name)
    output image (prefix_deformed.nii)
warp_field: (an existing file name)
    warp field (prefix_Warp.nii)

```

antsIntroduction[Link to code](#)Wraps command **antsIntroduction.sh**

Uses ANTS to generate matrices to warp data from one space to another.

Examples

```

>>> from nipy.interfaces.ants.legacy import antsIntroduction
>>> warp = antsIntroduction()

```



```
>>> warp.inputs.reference_image = 'Template_6.nii'
>>> warp.inputs.input_image = 'structural.nii'
>>> warp.inputs.max_iterations = [30,90,20]
>>> warp.cmdline
'antsIntroduction.sh -d 3 -i structural.nii -m 30x90x20 -o ants_ -r Template_6.
↪nii -t GR'
```

Inputs:

```
[Mandatory]
input_image: (an existing file name)
    input image to warp to template
    flag: -i %s
reference_image: (an existing file name)
    template file to warp to
    flag: -r %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bias_field_correction: (a boolean)
    Applies bias field correction to moving image
    flag: -n 1
dimension: (3 or 2, nipyte default value: 3)
    image dimension (2 or 3)
    flag: -d %d, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
force_proceed: (a boolean)
    force script to proceed even if headers may be incompatible
    flag: -f 1
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inverse_warp_template_labels: (a boolean)
    Applies inverse warp to the template labels to estimate label
    positions in target space (use for template-based segmentation)
    flag: -l
max_iterations: (a list of items which are an integer (int or long))
    maximum number of iterations (must be list of integers in the form
    [J,K,L...]: J = coarsest resolution iterations, K = middle
    resolution iterations, L = fine resolution iterations
    flag: -m %s
num_threads: (an integer (int or long), nipyte default value: 1)
    Number of ITK threads to use
out_prefix: (a unicode string, nipyte default value: ants_)
    Prefix that is prepended to all output files (default = ants_)
    flag: -o %s
quality_check: (a boolean)
    Perform a quality check of the result
    flag: -q 1
similarity_metric: ('PR' or 'CC' or 'MI' or 'MSQ')
    Type of similartiy metric used for registration (CC = cross
    correlation, MI = mutual information, PR = probability mapping, MSQ
    = mean square difference)
```

```

    flag: -s %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformation_model: ('GR' or 'EL' or 'SY' or 'S2' or 'EX' or 'DD'
    or 'RI' or 'RA', nipy default value: GR)
    Type of transofmrtaion model used for registration (EL = elastic
    transformation model, SY = SyN with time, arbitrary number of time
    points, S2 = SyN with time optimized for 2 time points, GR = greedy
    SyN, EX = exponential, DD = diffeomorphic demons style exponential
    mapping, RI = purely rigid, RA = affine rigid
    flag: -t %s

```

Outputs:

```

affine_transformation: (an existing file name)
    affine (prefix_Affine.txt)
input_file: (an existing file name)
    input image (prefix_repaired.nii)
inverse_warp_field: (an existing file name)
    inverse warp field (prefix_InverseWarp.nii)
output_file: (an existing file name)
    output image (prefix_deformed.nii)
warp_field: (an existing file name)
    warp field (prefix_Warp.nii)

```

buildtemplateparallel[Link to code](#)Wraps command **buildtemplateparallel.sh**

Generate a optimal average template

Warning: This can take a VERY long time to complete**Examples**

```

>>> from nipy.interfaces.ants.legacy import buildtemplateparallel
>>> tmp1 = buildtemplateparallel()
>>> tmp1.inputs.in_files = ['T1.nii', 'structural.nii']
>>> tmp1.inputs.max_iterations = [30, 90, 20]
>>> tmp1.cmdline
'buildtemplateparallel.sh -d 3 -i 4 -m 30x90x20 -o antsTMPL_ -c 0 -t GR T1.nii_
↳structural.nii'

```

Inputs:

```

[Mandatory]
in_files: (a list of items which are an existing file name)
    list of images to generate template from
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s

```

```

bias_field_correction: (a boolean)
    Applies bias field correction to moving image
    flag: -n 1
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    flag: -d %d, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gradient_step_size: (a float)
    smaller magnitude results in more cautious steps (default = .25)
    flag: -g %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
iteration_limit: (an integer (int or long), nipy default value: 4)
    iterations of template construction
    flag: -i %d
max_iterations: (a list of items which are an integer (int or long))
    maximum number of iterations (must be list of integers in the form
    [J,K,L...]: J = coarsest resolution iterations, K = middle
    resolution iterations, L = fine resolution iterations
    flag: -m %s
num_cores: (an integer (int or long))
    Requires parallelization = 2 (PEXEC). Sets number of cpu cores to
    use
    flag: -j %d
    requires: parallelization
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_prefix: (a unicode string, nipy default value: antsTMPL_)
    Prefix that is prepended to all output files (default = antsTMPL_)
    flag: -o %s
parallelization: (0 or 1 or 2, nipy default value: 0)
    control for parallel processing (0 = serial, 1 = use PBS, 2 = use
    PEXEC, 3 = use Apple XGrid
    flag: -c %d
rigid_body_registration: (a boolean)
    registers inputs before creating template (useful if no initial
    template available)
    flag: -r 1
similarity_metric: ('PR' or 'CC' or 'MI' or 'MSQ')
    Type of similarity metric used for registration (CC = cross
    correlation, MI = mutual information, PR = probability mapping, MSQ
    = mean square difference)
    flag: -s %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformation_model: ('GR' or 'EL' or 'SY' or 'S2' or 'EX' or 'DD',
    nipy default value: GR)
    Type of transformation model used for registration (EL = elastic
    transformation model, SY = SyN with time, arbitrary number of time
    points, S2 = SyN with time optimized for 2 time points, GR = greedy
    SyN, EX = exponential, DD = diffeomorphic demons style exponential
    mapping

```

```

    flag: -t %s
    use_first_as_target: (a boolean)
        uses first volume as target of all inputs. When not used, an
        unbiased average image is used to start.

```

Outputs:

```

    final_template_file: (an existing file name)
        final ANTS template
    subject_outfiles: (a list of items which are an existing file name)
        Outputs for each input image. Includes warp field, inverse warp,
        Affine, original image (repaired) and warped image (deformed)
    template_files: (a list of items which are an existing file name)
        Templates from different stages of iteration

```

interfaces.ants.registration**Registration**[Link to code](#)Wraps command **antsRegistration****Examples**

```

>>> import copy, pprint
>>> from nipy.interfaces.ants import Registration
>>> reg = Registration()
>>> reg.inputs.fixed_image = 'fixed1.nii'
>>> reg.inputs.moving_image = 'moving1.nii'
>>> reg.inputs.output_transform_prefix = "output_"
>>> reg.inputs.initial_moving_transform = 'trans.mat'
>>> reg.inputs.transforms = ['Affine', 'SyN']
>>> reg.inputs.transform_parameters = [(2.0,), (0.25, 3.0, 0.0)]
>>> reg.inputs.number_of_iterations = [[1500, 200], [100, 50, 30]]
>>> reg.inputs.dimension = 3
>>> reg.inputs.write_composite_transform = True
>>> reg.inputs.collapse_output_transforms = False
>>> reg.inputs.initialize_transforms_per_stage = False
>>> reg.inputs.metric = ['Mattes']*2
>>> reg.inputs.metric_weight = [1]*2 # Default (value ignored currently by ANTs)
>>> reg.inputs.radius_or_number_of_bins = [32]*2
>>> reg.inputs.sampling_strategy = ['Random', None]
>>> reg.inputs.sampling_percentage = [0.05, None]
>>> reg.inputs.convergence_threshold = [1.e-8, 1.e-9]
>>> reg.inputs.convergence_window_size = [20]*2
>>> reg.inputs.smoothing_sigmas = [[1,0], [2,1,0]]
>>> reg.inputs.sigma_units = ['vox'] * 2
>>> reg.inputs.shrink_factors = [[2,1], [3,2,1]]
>>> reg.inputs.use_estimate_learning_rate_once = [True, True]
>>> reg.inputs.use_histogram_matching = [True, True] # This is the default
>>> reg.inputs.output_warped_image = 'output_warped_image.nii.gz'
>>> reg.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↪moving-transform [ trans.mat, 0 ] --initialize-transforms-per-stage 0 --
↪interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
↪transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↪ 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↪shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↪1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1,
↪ 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox
↪shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-
↪matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-
↪transform 1 '

```

```
>>> reg.run()
```

```
>>> reg.inputs.invert_initial_moving_transform = True
>>> reg1 = copy.deepcopy(reg)
>>> reg1.inputs.winsorize_lower_quantile = 0.025
>>> reg1.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↳moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↳interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
↳transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↳ 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↳shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↳1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1,
↳ 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox -
↳shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-
↳matching 1 --winsorize-image-intensities [ 0.025, 1.0 ] --write-composite-
↳transform 1'
>>> reg1.run()
```

```
>>> reg2 = copy.deepcopy(reg)
>>> reg2.inputs.winsorize_upper_quantile = 0.975
>>> reg2.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↳moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↳interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
↳transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↳ 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↳shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↳1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1,
↳ 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox -
↳shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-
↳matching 1 --winsorize-image-intensities [ 0.0, 0.975 ] --write-composite-
↳transform 1'
```

```
>>> reg3 = copy.deepcopy(reg)
>>> reg3.inputs.winsorize_lower_quantile = 0.025
>>> reg3.inputs.winsorize_upper_quantile = 0.975
>>> reg3.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↳moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↳interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
↳transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↳ 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↳shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↳1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1,
↳ 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox -
↳shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-
↳matching 1 --winsorize-image-intensities [ 0.025, 0.975 ] --write-composite-
↳transform 1'
```

```
>>> reg3a = copy.deepcopy(reg)
>>> reg3a.inputs.float = True
>>> reg3a.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --float 1 --
↳initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↳interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
↳transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↳ 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↳shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↳1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1,
↳ 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox -
↳shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-
↳matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-
↳transform 1'
```

55.2. interfaces.ants.registration, 671

```
>>> reg3a.run()
```

```
>>> reg3b = copy.deepcopy(reg)
>>> reg3b.inputs.float = False
>>> reg3b.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --float 0 --
↳initial-moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↳interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
↳transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↳0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↳shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↳1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1,
↳32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox -
↳shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-
↳matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-
↳transform 1'
```

```
>>> # Test collapse transforms flag
>>> reg4 = copy.deepcopy(reg)
>>> reg4.inputs.save_state = 'trans.mat'
>>> reg4.inputs.restore_state = 'trans.mat'
>>> reg4.inputs.initialize_transforms_per_stage = True
>>> reg4.inputs.collapse_output_transforms = True
>>> outputs = reg4._list_outputs()
>>> pprint.pprint(outputs)
{'composite_transform': '../nipyte/testing/data/output_Composite.h5',
 'forward_invert_flags': [],
 'forward_transforms': [],
 'inverse_composite_transform': '../nipyte/testing/data/output_InverseComposite.
↳h5',
 'inverse_warped_image': <undefined>,
 'reverse_invert_flags': [],
 'reverse_transforms': [],
 'save_state': '../nipyte/testing/data/trans.mat',
 'warped_image': '../nipyte/testing/data/output_warped_image.nii.gz'}
>>> reg4.cmdline
'antsRegistration --collapse-output-transforms 1 --dimensionality 3 --initial-
↳moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 1 --
↳interpolation Linear --output [ output_, output_warped_image.nii.gz ] --restore-
↳state trans.mat --save-state trans.mat --transform Affine[ 2.0 ] --metric_
↳Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200,
↳1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --shrink-factors 2x1 --use-estimate-
↳learning-rate-once 1 --use-histogram-matching 1 --transform SyN[ 0.25, 3.0, 0.0_
↳] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32 ] --convergence [ 100x50x30, _
↳1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-
↳estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-
↳intensities [ 0.0, 1.0 ] --write-composite-transform 1'
```

```
>>> # Test collapse transforms flag
>>> reg4b = copy.deepcopy(reg4)
>>> reg4b.inputs.write_composite_transform = False
>>> outputs = reg4b._list_outputs()
>>> pprint.pprint(outputs)
{'composite_transform': <undefined>,
 'forward_invert_flags': [False, False],
 'forward_transforms': ['../nipyte/testing/data/output_0GenericAffine.mat',
 '.../nipyte/testing/data/output_1Warp.nii.gz'],
 'inverse_composite_transform': <undefined>,'
```

```

'inverse_warped_image': <undefined>,
'reverse_invert_flags': [True, False],
'reverse_transforms': ['.../nipy/testing/data/output_0GenericAffine.mat', '.../
↳nipy/testing/data/output_1InverseWarp.nii.gz'],
'save_state': '.../nipy/testing/data/trans.mat',
'warped_image': '.../nipy/testing/data/output_warped_image.nii.gz'}
>>> reg4b.aggregate_outputs()
>>> reg4b.cmdline
'antsRegistration --collapse-output-transforms 1 --dimensionality 3 --initial-
↳moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 1 --
↳interpolation Linear --output [ output_, output_warped_image.nii.gz ] --restore-
↳state trans.mat --save-state trans.mat --transform Affine[ 2.0 ] --metric_
↳Mattes[ fixed1.nii, moving1.nii, 1, 32, Random, 0.05 ] --convergence [ 1500x200,
↳1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --shrink-factors 2x1 --use-estimate-
↳learning-rate-once 1 --use-histogram-matching 1 --transform SyN[ 0.25, 3.0, 0.0_
↳] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32 ] --convergence [ 100x50x30,_
↳1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-
↳estimate-learning-rate-once 1 --use-histogram-matching 1 --winsorize-image-
↳intensities [ 0.0, 1.0 ] --write-composite-transform 0'

```

```

>>> # Test multiple metrics per stage
>>> reg5 = copy.deepcopy(reg)
>>> reg5.inputs.fixed_image = 'fixed1.nii'
>>> reg5.inputs.moving_image = 'moving1.nii'
>>> reg5.inputs.metric = ['Mattes', ['Mattes', 'CC']]
>>> reg5.inputs.metric_weight = [1, [.5,.5]]
>>> reg5.inputs.radius_or_number_of_bins = [32, [32, 4] ]
>>> reg5.inputs.sampling_strategy = ['Random', None] # use default strategy in_
↳second stage
>>> reg5.inputs.sampling_percentage = [0.05, [0.05, 0.10]]
>>> reg5.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↳moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↳interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
↳transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↳0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↳shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↳1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 0.
↳5, 32, None, 0.05 ] --metric CC[ fixed1.nii, moving1.nii, 0.5, 4, None, 0.1 ] --
↳convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --shrink-
↳factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching 1 --
↳winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-transform 1'

```

```

>>> # Test multiple inputs
>>> reg6 = copy.deepcopy(reg5)
>>> reg6.inputs.fixed_image = ['fixed1.nii', 'fixed2.nii']
>>> reg6.inputs.moving_image = ['moving1.nii', 'moving2.nii']
>>> reg6.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↳moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↳interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
↳transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↳0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↳shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↳1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 0.
↳5, 32, None, 0.05 ] --metric CC[ fixed2.nii, moving2.nii, 0.5, 4, None, 0.1 ] --
↳convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox --shrink-
↳factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching 1 --
↳winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-transform 1'

```

```
>>> # Test Interpolation Parameters (BSpline)
>>> reg7a = copy.deepcopy(reg)
>>> reg7a.inputs.interpolation = 'BSpline'
>>> reg7a.inputs.interpolation_parameters = (3,)
>>> reg7a.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↪moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↪interpolation BSpline[ 3 ] --output [ output_, output_warped_image.nii.gz ] --
↪transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↪0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↪shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↪1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1,
↪32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.0x0.0vox -
↪shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-histogram-
↪matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-composite-
↪transform 1'
```

```
>>> # Test Interpolation Parameters (MultiLabel/Gaussian)
>>> reg7b = copy.deepcopy(reg)
>>> reg7b.inputs.interpolation = 'Gaussian'
>>> reg7b.inputs.interpolation_parameters = (1.0, 1.0)
>>> reg7b.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↪moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↪interpolation Gaussian[ 1.0, 1.0 ] --output [ output_, output_warped_image.nii.
↪gz ] --transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32,
↪Random, 0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.
↪0vox --shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-
↪matching 1 --transform SyN[ 0.25, 3.0, 0.0 ] --metric Mattes[ fixed1.nii,
↪moving1.nii, 1, 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas_
↪2.0x1.0x0.0vox --shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-
↪histogram-matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-
↪composite-transform 1'
```

```
>>> # Test Extended Transform Parameters
>>> reg8 = copy.deepcopy(reg)
>>> reg8.inputs.transforms = ['Affine', 'BSplineSyN']
>>> reg8.inputs.transform_parameters = [(2.0,), (0.25, 26, 0, 3)]
>>> reg8.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↪moving-transform [ trans.mat, 1 ] --initialize-transforms-per-stage 0 --
↪interpolation Linear --output [ output_, output_warped_image.nii.gz ] --
↪transform Affine[ 2.0 ] --metric Mattes[ fixed1.nii, moving1.nii, 1, 32, Random,
↪0.05 ] --convergence [ 1500x200, 1e-08, 20 ] --smoothing-sigmas 1.0x0.0vox --
↪shrink-factors 2x1 --use-estimate-learning-rate-once 1 --use-histogram-matching_
↪1 --transform BSplineSyN[ 0.25, 26, 0, 3 ] --metric Mattes[ fixed1.nii, moving1.
↪nii, 1, 32 ] --convergence [ 100x50x30, 1e-09, 20 ] --smoothing-sigmas 2.0x1.
↪0x0.0vox --shrink-factors 3x2x1 --use-estimate-learning-rate-once 1 --use-
↪histogram-matching 1 --winsorize-image-intensities [ 0.0, 1.0 ] --write-
↪composite-transform 1'
```

Inputs:

```
[Mandatory]
fixed_image: (a list of items which are an existing file name)
              image to apply transformation to (generally a coregistered
```



```

functional)
metric: (a list of items which are 'CC' or 'MeanSquares' or 'Demons'
        or 'GC' or 'MI' or 'Mattes' or a list of items which are 'CC' or
        'MeanSquares' or 'Demons' or 'GC' or 'MI' or 'Mattes')
        the metric(s) to use for each stage. Note that multiple metrics per
        stage are not supported in ANTS 1.9.1 and earlier.
metric_weight: (a list of items which are a float or a list of items
                which are a float, nipy default value: [1.0])
                the metric weight(s) for each stage. The weights must sum to 1 per
                stage.
                requires: metric
moving_image: (a list of items which are an existing file name)
                image to apply transformation to (generally a coregistered
                functional)
shrink_factors: (a list of items which are a list of items which are
                 an integer (int or long))
smoothing_sigmas: (a list of items which are a list of items which
                   are a float)
transforms: (a list of items which are 'Rigid' or 'Affine' or
            'CompositeAffine' or 'Similarity' or 'Translation' or 'BSpline' or
            'GaussianDisplacementField' or 'TimeVaryingVelocityField' or
            'TimeVaryingBSplineVelocityField' or 'SyN' or 'BSplineSyN' or
            'Exponential' or 'BSplineExponential')
flag: %s

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
collapse_output_transforms: (a boolean, nipy default value: False)
        Collapse output transforms. Specifically, enabling this option
        combines all adjacent linear transforms and composes all adjacent
        displacement field transforms before writing the results to disk.
        flag: --collapse-output-transforms %d
convergence_threshold: (a list of at least 1 items which are a float,
                       nipy default value: [1e-06])
                       requires: number_of_iterations
convergence_window_size: (a list of at least 1 items which are an
                          integer (int or long), nipy default value: [10])
                          requires: convergence_threshold
dimension: (3 or 2, nipy default value: 3)
            image dimension (2 or 3)
            flag: --dimensionality %d
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
fixed_image_mask: (an existing file name)
                  mask used to limit metric sampling region of the fixed image
                  flag: %s
float: (a boolean)
        Use float instead of double for computations.
        flag: --float %d
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
initial_moving_transform: (an existing file name)
                          flag: %s

```

```

    mutually_exclusive: initial_moving_transform_com
initial_moving_transform_com: (0 or 1 or 2)
    Use center of mass for moving transform
    flag: %s
    mutually_exclusive: initial_moving_transform
initialize_transforms_per_stage: (a boolean, nipy default value:
    False)
    Initialize linear transforms from the previous stage. By enabling
    this option, the current linear stage transform is directly
    initialized from the previous stages linear transform; this allows
    multiple linear stages to be run where each stage directly updates
    the estimated linear transform from the previous stage. (e.g.
    Translation -> Rigid -> Affine).
    flag: --initialize-transforms-per-stage %d
interpolation: ('Linear' or 'NearestNeighbor' or 'CosineWindowedSinc'
    or 'WelchWindowedSinc' or 'HammingWindowedSinc' or
    'LanczosWindowedSinc' or 'BSpline' or 'MultiLabel' or 'Gaussian',
    nipy default value: Linear)
    flag: %s
interpolation_parameters: (a tuple of the form: (an integer (int or
    long)) or a tuple of the form: (a float, a float))
invert_initial_moving_transform: (a boolean)
    mutually_exclusive: initial_moving_transform_com
    requires: initial_moving_transform
metric_item_trait: ('CC' or 'MeanSquares' or 'Demons' or 'GC' or 'MI'
    or 'Mattes')
metric_stage_trait: ('CC' or 'MeanSquares' or 'Demons' or 'GC' or
    'MI' or 'Mattes' or a list of items which are 'CC' or 'MeanSquares'
    or 'Demons' or 'GC' or 'MI' or 'Mattes')
metric_weight_item_trait: (a float)
metric_weight_stage_trait: (a float or a list of items which are a
    float)
moving_image_mask: (an existing file name)
    mask used to limit metric sampling region of the moving image
    requires: fixed_image_mask
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
number_of_iterations: (a list of items which are a list of items
    which are an integer (int or long))
output_inverse_warped_image: (a boolean or a file name)
    requires: output_warped_image
output_transform_prefix: (a unicode string, nipy default value:
    transform)
    flag: %s
output_warped_image: (a boolean or a file name)
radius_bins_item_trait: (an integer (int or long))
radius_bins_stage_trait: (an integer (int or long) or a list of items
    which are an integer (int or long))
radius_or_number_of_bins: (a list of items which are an integer (int
    or long) or a list of items which are an integer (int or long),
    nipy default value: [5])
    the number of bins in each stage for the MI and Mattes metric, the
    radius for other metrics
    requires: metric_weight
restore_state: (an existing file name)
    Filename for restoring the internal restorable state of the
    registration
    flag: --restore-state %s

```

```

sampling_percentage: (a list of items which are 0.0 <= a floating
    point number <= 1.0 or None or a list of items which are 0.0 <= a
    floating point number <= 1.0 or None)
    the metric sampling percentage(s) to use for each stage
    requires: sampling_strategy
sampling_percentage_item_trait: (0.0 <= a floating point number <=
    1.0 or None)
sampling_percentage_stage_trait: (0.0 <= a floating point number <=
    1.0 or None or a list of items which are 0.0 <= a floating point
    number <= 1.0 or None)
sampling_strategy: (a list of items which are 'None' or 'Regular' or
    'Random' or None or a list of items which are 'None' or 'Regular'
    or 'Random' or None)
    the metric sampling strategy (strategies) for each stage
    requires: metric_weight
sampling_strategy_item_trait: ('None' or 'Regular' or 'Random' or
    None)
sampling_strategy_stage_trait: ('None' or 'Regular' or 'Random' or
    None or a list of items which are 'None' or 'Regular' or 'Random'
    or None)
save_state: (a file name)
    Filename for saving the internal restorable state of the
    registration
    flag: --save-state %s
sigma_units: (a list of items which are 'mm' or 'vox')
    units for smoothing sigmas
    requires: smoothing_sigmas
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transform_parameters: (a list of items which are a tuple of the form:
    (a float) or a tuple of the form: (a float, a float, a float) or a
    tuple of the form: (a float, an integer (int or long), an integer
    (int or long), an integer (int or long)) or a tuple of the form: (a
    float, an integer (int or long), a float, a float, a float, a
    float) or a tuple of the form: (a float, a float, a float, an
    integer (int or long)) or a tuple of the form: (a float, an integer
    (int or long), an integer (int or long), an integer (int or long),
    an integer (int or long)))
use_estimate_learning_rate_once: (a list of items which are a
    boolean)
use_histogram_matching: (a boolean or a list of items which are a
    boolean, nipy default value: True)
verbose: (a boolean)
    flag: -v
winsorize_lower_quantile: (0.0 <= a floating point number <= 1.0,
    nipy default value: 0.0)
    The Lower quantile to clip image ranges
    flag: %s
winsorize_upper_quantile: (0.0 <= a floating point number <= 1.0,
    nipy default value: 1.0)
    The Upper quantile to clip image ranges
    flag: %s
write_composite_transform: (a boolean, nipy default value: False)
    flag: --write-composite-transform %d

```

Outputs:

```

composite_transform: (an existing file name)
    Composite transform file
forward_invert_flags: (a list of items which are a boolean)
    List of flags corresponding to the forward transforms
forward_transforms: (a list of items which are an existing file name)
    List of output transforms for forward registration
inverse_composite_transform: (a file name)
    Inverse composite transform file
inverse_warped_image: (a file name)
    Outputs the inverse of the warped image
reverse_invert_flags: (a list of items which are a boolean)
    List of flags corresponding to the reverse transforms
reverse_transforms: (a list of items which are an existing file name)
    List of output transforms for reverse registration
save_state: (a file name)
    The saved registration state to be restored
warped_image: (a file name)
    Outputs warped image

```

interfaces.ants.resampling

ApplyTransforms

[Link to code](#)

Wraps command **antsApplyTransforms**

ApplyTransforms, applied to an input image, transforms it according to a reference image and a transform (or a set of transforms).

Examples

```

>>> from nipy.interfaces.ants import ApplyTransforms
>>> at = ApplyTransforms()
>>> at.inputs.dimension = 3
>>> at.inputs.input_image = 'moving1.nii'
>>> at.inputs.reference_image = 'fixed1.nii'
>>> at.inputs.output_image = 'deformed_moving1.nii'
>>> at.inputs.interpolation = 'Linear'
>>> at.inputs.default_value = 0
>>> at.inputs.transforms = ['ants_Warp.nii.gz', 'trans.mat']
>>> at.inputs.invert_transform_flags = [False, False]
>>> at.cmdline
'antsApplyTransforms --default-value 0 --dimensionality 3 --input moving1.nii --
↪interpolation Linear --output deformed_moving1.nii --reference-image fixed1.nii_
↪--transform [ ants_Warp.nii.gz, 0 ] --transform [ trans.mat, 0 ]'

```

```

>>> at1 = ApplyTransforms()
>>> at1.inputs.dimension = 3
>>> at1.inputs.input_image = 'moving1.nii'
>>> at1.inputs.reference_image = 'fixed1.nii'
>>> at1.inputs.output_image = 'deformed_moving1.nii'
>>> at1.inputs.interpolation = 'BSpline'
>>> at1.inputs.interpolation_parameters = (5,)
>>> at1.inputs.default_value = 0
>>> at1.inputs.transforms = ['ants_Warp.nii.gz', 'trans.mat']
>>> at1.inputs.invert_transform_flags = [False, False]

```

```
>>> atl.cmdline
'antsApplyTransforms --default-value 0 --dimensionality 3 --input movingl.nii --
↳ interpolation BSpline[ 5 ] --output deformed_movingl.nii --reference-image_
↳ fixedl.nii --transform [ ants_Warp.nii.gz, 0 ] --transform [ trans.mat, 0 ]'
```

Inputs:

```
[Mandatory]
input_image: (an existing file name)
    image to apply transformation to (generally a coregistered
    functional)
    flag: --input %s
reference_image: (an existing file name)
    reference image space that you wish to warp INTO
    flag: --reference-image %s
transforms: (a list of items which are an existing file name)
    transform files: will be applied in reverse order. For example, the
    last specified transform will be applied first.
    flag: %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
default_value: (a float, nipy default value: 0.0)
    flag: --default-value %g
dimension: (2 or 3 or 4)
    This option forces the image to be treated as a specified-
    dimensional image. If not specified, antsWarp tries to infer the
    dimensionality from the input image.
    flag: --dimensionality %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
float: (a boolean)
    Use float instead of double for computations.
    flag: --float %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
input_image_type: (0 or 1 or 2 or 3)
    Option specifying the input image type of scalar (default), vector,
    tensor, or time series.
    flag: --input-image-type %d
interpolation: ('Linear' or 'NearestNeighbor' or 'CosineWindowedSinc'
    or 'WelchWindowedSinc' or 'HammingWindowedSinc' or
    'LanczosWindowedSinc' or 'MultiLabel' or 'Gaussian' or 'BSpline',
    nipy default value: Linear)
    flag: %s
interpolation_parameters: (a tuple of the form: (an integer (int or
    long)) or a tuple of the form: (a float, a float))
invert_transform_flags: (a list of items which are a boolean)
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_postfix: (a unicode string, nipy default value: _trans)
    Postfix that is appended to all output files (default = _trans)
output_image: (a unicode string)
```

```

        output file name
        flag: --output %s
print_out_composite_warp_file: (a boolean)
        output a composite warp file instead of a transformed image
        requires: output_image
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

output_image: (an existing file name)
        Warped image

```

ApplyTransformsToPoints

[Link to code](#)**Wraps command `antsApplyTransformsToPoints`**

ApplyTransformsToPoints, applied to an CSV file, transforms coordinates using provided transform (or a set of transforms).

Examples

```

>>> from nipy.interfaces.ants import ApplyTransforms
>>> at = ApplyTransformsToPoints()
>>> at.inputs.dimension = 3
>>> at.inputs.input_file = 'moving.csv'
>>> at.inputs.transforms = ['trans.mat', 'ants_Warp.nii.gz']
>>> at.inputs.invert_transform_flags = [False, False]
>>> at.cmdline
'antsApplyTransformsToPoints --dimensionality 3 --input moving.csv --output_
↪moving_transformed.csv --transform [ trans.mat, 0 ] --transform [ ants_Warp.nii.
↪gz, 0 ]'

```

Inputs:

```

[Mandatory]
input_file: (an existing file name)
        Currently, the only input supported is a csv file with columns
        including x,y (2D), x,y,z (3D) or x,y,z,t,label (4D) column
        headers.The points should be defined in physical space.If in doubt
        how to convert coordinates from your files to the spacerequired by
        antsApplyTransformsToPoints try creating/drawing a simplelabel
        volume with only one voxel set to 1 and all others set to 0.Write
        down the voxel coordinates. Then use ImageMaths LabelStats to
        findout what coordinates for this voxel antsApplyTransformsToPoints
        isexpecting.
        flag: --input %s
transforms: (a list of items which are an existing file name)
        transforms that will be applied to the points
        flag: %s

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s

```

```

dimension: (2 or 3 or 4)
    This option forces the image to be treated as a specified-
    dimensional image. If not specified, antsWarp tries to infer the
    dimensionality from the input image.
    flag: --dimensionality %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert_transform_flags: (a list of items which are a boolean)
    list indicating if a transform should be reversed
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
output_file: (a unicode string)
    Name of the output CSV file
    flag: --output %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

output_file: (an existing file name)
    csv file with transformed coordinates

```

WarpImageMultiTransform[Link to code](#)Wraps command **WarpImageMultiTransform**

Warpes an image from one space to another

Examples

```

>>> from nipy.interfaces.ants import WarpImageMultiTransform
>>> wimt = WarpImageMultiTransform()
>>> wimt.inputs.input_image = 'structural.nii'
>>> wimt.inputs.reference_image = 'ants_deformed.nii.gz'
>>> wimt.inputs.transformation_series = ['ants_Warp.nii.gz', 'ants_Affine.txt']
>>> wimt.cmdline
'WarpImageMultiTransform 3 structural.nii structural_wimt.nii -R ants_deformed.
↪nii.gz ants_Warp.nii.gz ants_Affine.txt'

```

```

>>> wimt = WarpImageMultiTransform()
>>> wimt.inputs.input_image = 'diffusion_weighted.nii'
>>> wimt.inputs.reference_image = 'functional.nii'
>>> wimt.inputs.transformation_series = ['func2anat_coreg_Affine.txt', 'func2anat_
↪InverseWarp.nii.gz', 'dwi2anat_Warp.nii.gz', 'dwi2anat_coreg_Affine.txt']
>>> wimt.inputs.invert_affine = [1]
>>> wimt.cmdline
'WarpImageMultiTransform 3 diffusion_weighted.nii diffusion_weighted_wimt.nii -R_
↪functional.nii -i func2anat_coreg_Affine.txt func2anat_InverseWarp.nii.gz_
↪dwi2anat_Warp.nii.gz dwi2anat_coreg_Affine.txt'

```

Inputs:

```

[Mandatory]
input_image: (a file name)
    image to apply transformation to (generally a coregistered
    functional)
    flag: %s, position: 2
transformation_series: (a list of items which are an existing file
    name)
    transformation file(s) to be applied
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    flag: %d, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert_affine: (a list of items which are an integer (int or long))
    List of Affine transformations to invert.E.g.: [1,4,5] inverts the
    1st, 4th, and 5th Affines found in transformation_series. Note that
    indexing starts with 1 and does not include warp fields. Affine
    transformations are distinguished from warp fields by the word
    "affine" included in their filenames.
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_postfix: (a file name, nipy default value: _wimt)
    Postfix that is prepended to all output files (default = _wimt)
    mutually_exclusive: output_image
output_image: (a file name)
    name of the output warped image
    flag: %s, position: 3
    mutually_exclusive: out_postfix
reference_image: (a file name)
    reference image space that you wish to warp INTO
    flag: -R %s
    mutually_exclusive: tightest_box
reslice_by_header: (a boolean)
    Uses orientation matrix and origin encoded in reference image file
    header. Not typically used with additional transforms
    flag: --reslice-by-header
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tightest_box: (a boolean)
    computes tightest bounding box (overridden by reference_image if
    given)
    flag: --tightest-bounding-box
    mutually_exclusive: reference_image
use_bspline: (a boolean)

```



```

        Use 3rd order B-Spline interpolation
        flag: --use-BSpline
    use_nearest: (a boolean)
        Use nearest neighbor interpolation
        flag: --use-NN

```

Outputs:

```

output_image: (an existing file name)
    Warped image

```

WarpTimeSeriesImageMultiTransform[Link to code](#)Wraps command **WarpTimeSeriesImageMultiTransform**

Warp a time-series from one space to another

Examples

```

>>> from nipy.interfaces.ants import WarpTimeSeriesImageMultiTransform
>>> wtsimt = WarpTimeSeriesImageMultiTransform()
>>> wtsimt.inputs.input_image = 'resting.nii'
>>> wtsimt.inputs.reference_image = 'ants_deformed.nii.gz'
>>> wtsimt.inputs.transformation_series = ['ants_Warp.nii.gz', 'ants_Affine.txt']
>>> wtsimt.cmdline
'WarpTimeSeriesImageMultiTransform 4 resting.nii resting_wtsimt.nii -R ants_
↳deformed.nii.gz ants_Warp.nii.gz ants_Affine.txt'

```

Inputs:

```

[Mandatory]
input_image: (a file name)
    image to apply transformation to (generally a coregistered
    functional)
    flag: %s
transformation_series: (a list of items which are an existing file
    name)
    transformation file(s) to be applied
    flag: %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dimension: (4 or 3, nipy default value: 4)
    image dimension (3 or 4)
    flag: %d, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert_affine: (a list of items which are an integer (int or long))
    List of Affine transformations to invert. E.g.: [1,4,5] inverts the
    1st, 4th, and 5th Affines found in transformation_series

```

```

num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_postfix: (a unicode string, nipy default value: _wtsimt)
    Postfix that is prepended to all output files (default = _wtsimt)
    flag: %s
reference_image: (a file name)
    reference image space that you wish to warp INTO
    flag: -R %s
    mutually_exclusive: tightest_box
reslice_by_header: (a boolean)
    Uses orientation matrix and origin encoded in reference image file
    header. Not typically used with additional transforms
    flag: --reslice-by-header
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tightest_box: (a boolean)
    computes tightest bounding box (overrided by reference_image if
    given)
    flag: --tightest-bounding-box
    mutually_exclusive: reference_image
use_bspline: (a boolean)
    Use 3rd order B-Spline interpolation
    flag: --use-Bspline
use_nearest: (a boolean)
    Use nearest neighbor interpolation
    flag: --use-NN

```

Outputs:

```

output_image: (an existing file name)
    Warped image

```

interfaces.ants.segmentation

AntsJointFusion

[Link to code](#)Wraps command **antsJointFusion**

Examples

```

>>> from nipy.interfaces.ants import AntsJointFusion
>>> antsjointfusion = AntsJointFusion()
>>> antsjointfusion.inputs.out_label_fusion = 'ants_fusion_label_output.nii'
>>> antsjointfusion.inputs.atlas_image = [ ['rcls1.nii', 'rcls2.nii'] ]
>>> antsjointfusion.inputs.atlas_segmentation_image = ['segmentation0.nii.gz']
>>> antsjointfusion.inputs.target_image = ['im1.nii']
>>> antsjointfusion.cmdline
"antsJointFusion -a 0.1 -g ['rcls1.nii', 'rcls2.nii'] -l segmentation0.nii.gz -b_
↳2.0 -o ants_fusion_label_output.nii -s 3x3x3 -t ['im1.nii']"

```

```

>>> antsjointfusion.inputs.target_image = [ ['im1.nii', 'im2.nii'] ]
>>> antsjointfusion.cmdline
"antsJointFusion -a 0.1 -g ['rcls1.nii', 'rcls2.nii'] -l segmentation0.nii.gz -b_
↳2.0 -o ants_fusion_label_output.nii -s 3x3x3 -t ['im1.nii', 'im2.nii']"

```

```
>>> antsjointfusion.inputs.atlas_image = [ ['rc1s1.nii','rc1s2.nii'],
...                                         ['rc2s1.nii','rc2s2.nii'] ]
>>> antsjointfusion.inputs.atlas_segmentation_image = ['segmentation0.nii.gz',
...                                                    'segmentation1.nii.gz']
>>> antsjointfusion.cmdline
"antsJointFusion -a 0.1 -g ['rc1s1.nii', 'rc1s2.nii'] -g ['rc2s1.nii', 'rc2s2.nii
↪'] -l segmentation0.nii.gz -l segmentation1.nii.gz -b 2.0 -o ants_fusion_label_
↪output.nii -s 3x3x3 -t ['im1.nii', 'im2.nii']"
```

```
>>> antsjointfusion.inputs.dimension = 3
>>> antsjointfusion.inputs.alpha = 0.5
>>> antsjointfusion.inputs.beta = 1.0
>>> antsjointfusion.inputs.patch_radius = [3,2,1]
>>> antsjointfusion.inputs.search_radius = [3]
>>> antsjointfusion.cmdline
"antsJointFusion -a 0.5 -g ['rc1s1.nii', 'rc1s2.nii'] -g ['rc2s1.nii', 'rc2s2.nii
↪'] -l segmentation0.nii.gz -l segmentation1.nii.gz -b 1.0 -d 3 -o ants_fusion_
↪label_output.nii -p 3x2x1 -s 3 -t ['im1.nii', 'im2.nii']"
```

```
>>> antsjointfusion.inputs.search_radius = ['mask.nii']
>>> antsjointfusion.inputs.verbose = True
>>> antsjointfusion.inputs.exclusion_image = ['roi01.nii', 'roi02.nii']
>>> antsjointfusion.inputs.exclusion_image_label = ['1','2']
>>> antsjointfusion.cmdline
"antsJointFusion -a 0.5 -g ['rc1s1.nii', 'rc1s2.nii'] -g ['rc2s1.nii', 'rc2s2.nii
↪'] -l segmentation0.nii.gz -l segmentation1.nii.gz -b 1.0 -d 3 -e 1[roi01.nii] -
↪e 2[roi02.nii] -o ants_fusion_label_output.nii -p 3x2x1 -s mask.nii -t ['im1.nii
↪', 'im2.nii'] -v"
```

```
>>> antsjointfusion.inputs.out_label_fusion = 'ants_fusion_label_output.nii'
>>> antsjointfusion.inputs.out_intensity_fusion_name_format = 'ants_joint_fusion_
↪intensity_%d.nii.gz'
>>> antsjointfusion.inputs.out_label_post_prob_name_format = 'ants_joint_fusion_
↪posterior_%d.nii.gz'
>>> antsjointfusion.inputs.out_atlas_voting_weight_name_format = 'ants_joint_
↪fusion_voting_weight_%d.nii.gz'
>>> antsjointfusion.cmdline
"antsJointFusion -a 0.5 -g ['rc1s1.nii', 'rc1s2.nii'] -g ['rc2s1.nii', 'rc2s2.nii
↪'] -l segmentation0.nii.gz -l segmentation1.nii.gz -b 1.0 -d 3 -e 1[roi01.nii] -
↪e 2[roi02.nii] -o [ants_fusion_label_output.nii, ants_joint_fusion_intensity_
↪%d.nii.gz, ants_joint_fusion_posterior_%d.nii.gz, ants_joint_fusion_voting_
↪weight_%d.nii.gz] -p 3x2x1 -s mask.nii -t ['im1.nii', 'im2.nii'] -v"
```

Inputs:

```
[Mandatory]
atlas_image: (a list of items which are a list of items which are an
              existing file name)
              The atlas image (or multimodal atlas images) assumed to be aligned
              to a common image domain.
              flag: -g %s...
atlas_segmentation_image: (a list of items which are an existing file
                           name)
                           The atlas segmentation images. For performing label fusion the
                           number of specified segmentations should be identical to the number
                           of atlas image sets.
```

```

    flag: -l %s...
target_image: (a list of items which are a list of items which are an
    existing file name)
    The target image (or multimodal target images) assumed to be aligned
    to a common image domain.
    flag: -t %s

[Optional]
alpha: (a float, nipy default value: 0.1)
    Regularization term added to matrix Mx for calculating the inverse.
    Default = 0.1
    flag: -a %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
beta: (a float, nipy default value: 2.0)
    Exponent for mapping intensity difference to the joint error.
    Default = 2.0
    flag: -b %s
constrain_nonnegative: (a boolean, nipy default value: False)
    Constrain solution to non-negative weights.
    flag: -c
dimension: (3 or 2 or 4)
    This option forces the image to be treated as a specified-
    dimensional image. If not specified, the program tries to infer the
    dimensionality from the input image.
    flag: -d %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
exclusion_image: (a list of items which are an existing file name)
    Specify an exclusion region for the given label.
exclusion_image_label: (a list of items which are a unicode string)
    Specify a label for the exclusion region.
    flag: -e %s
    requires: exclusion_image
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_image: (an existing file name)
    If a mask image is specified, fusion is only performed in the mask
    region.
    flag: -x %s
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_atlas_voting_weight_name_format: (a unicode string)
    Optional atlas voting weight image file name format.
    requires: out_label_fusion, out_intensity_fusion_name_format,
    out_label_post_prob_name_format
out_intensity_fusion_name_format: (a unicode string)
    Optional intensity fusion image file name format.
out_label_fusion: (a file name)
    The output label fusion image.
    flag: %s
out_label_post_prob_name_format: (a unicode string)
    Optional label posterior probability image file name format.
    requires: out_label_fusion, out_intensity_fusion_name_format

```

```

patch_metric: ('PC' or 'MSQ')
    Metric to be used in determining the most similar neighborhood
    patch. Options include Pearson's correlation (PC) and mean squares
    (MSQ). Default = PC (Pearson correlation).
    flag: -m %s
patch_radius: (a list of items which are a value of class 'int')
    Patch radius for similarity measures.Default: 2x2x2
    flag: -p %s
retain_atlas_voting_images: (a boolean, nipyype default value: False)
    Retain atlas voting images. Default = false
    flag: -f
retain_label_posterior_images: (a boolean, nipyype default value:
    False)
    Retain label posterior probability images. Requires atlas
    segmentations to be specified. Default = false
    flag: -r
    requires: atlas_segmentation_image
search_radius: (a list of from 1 to 3 items which are any value,
    nipyype default value: [3, 3, 3])
    Search radius for similarity measures. Default = 3x3x3. One can also
    specify an image where the value at the voxel specifies the
    isotropic search radius at that voxel.
    flag: -s %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Verbose output.
    flag: -v

```

Outputs:

```

out_atlas_voting_weight_name_format: (a unicode string)
out_intensity_fusion_name_format: (a unicode string)
out_label_fusion: (an existing file name)
out_label_post_prob_name_format: (a unicode string)

```

Atropos

[Link to code](#)

Wraps command **Atropos**

A finite mixture modeling (FMM) segmentation approach with possibilities for specifying prior constraints. These prior constraints include the specification of a prior label image, prior probability images (one for each class), and/or an MRF prior to enforce spatial smoothing of the labels. Similar algorithms include FAST and SPM.

Examples

```

>>> from nipyype.interfaces.ants import Atropos
>>> at = Atropos()
>>> at.inputs.dimension = 3
>>> at.inputs.intensity_images = 'structural.nii'
>>> at.inputs.mask_image = 'mask.nii'
>>> at.inputs.initialization = 'PriorProbabilityImages'
>>> at.inputs.prior_probability_images = ['rc1s1.nii', 'rc1s2.nii']
>>> at.inputs.number_of_tissue_classes = 2

```

```

>>> at.inputs.prior_weighting = 0.8
>>> at.inputs.prior_probability_threshold = 0.0000001
>>> at.inputs.likelihood_model = 'Gaussian'
>>> at.inputs.mrf_smoothing_factor = 0.2
>>> at.inputs.mrf_radius = [1, 1, 1]
>>> at.inputs.icm_use_synchronous_update = True
>>> at.inputs.maximum_number_of_icm_terations = 1
>>> at.inputs.n_iterations = 5
>>> at.inputs.convergence_threshold = 0.000001
>>> at.inputs.posterior_formulation = 'Socrates'
>>> at.inputs.use_mixture_model_proportions = True
>>> at.inputs.save_posteriors = True
>>> at.cmdline
'Atropos --image-dimensionality 3 --icm [1,1] --initialization_
↪PriorProbabilityImages[2,priors/priorProbImages%02d.nii,0.8,1e-07] --intensity-
↪image structural.nii --likelihood-model Gaussian --mask-image mask.nii --mrf [0.
↪2,1x1x1] --convergence [5,1e-06] --output [structural_labeled.nii,POSTERIOR_
↪%02d.nii.gz] --posterior-formulation Socrates[1] --use-random-seed 1'

```

Inputs:

```

[Mandatory]
initialization: ('Random' or 'Otsu' or 'KMeans' or
                'PriorProbabilityImages' or 'PriorLabelImage')
                flag: %s
                requires: number_of_tissue_classes
intensity_images: (a list of items which are an existing file name)
                  flag: --intensity-image %s...
mask_image: (an existing file name)
             flag: --mask-image %s
number_of_tissue_classes: (an integer (int or long))

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
convergence_threshold: (a float)
                       requires: n_iterations
dimension: (3 or 2 or 4, nipy default value: 3)
            image dimension (2, 3, or 4)
            flag: --image-dimensionality %d
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
icm_use_synchronous_update: (a boolean)
                             flag: %s
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
likelihood_model: (a unicode string)
                  flag: --likelihood-model %s
maximum_number_of_icm_terations: (an integer (int or long))
                                  requires: icm_use_synchronous_update
mrf_radius: (a list of items which are an integer (int or long))
             requires: mrf_smoothing_factor
mrf_smoothing_factor: (a float)
                      flag: %s

```

```

n_iterations: (an integer (int or long))
    flag: %s
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_classified_image_name: (a file name)
    flag: %s
output_posteriors_name_template: (a unicode string, nipy default
    value: POSTERIOR_%02d.nii.gz)
posterior_formulation: (a unicode string)
    flag: %s
prior_probability_images: (a list of items which are an existing file
    name)
prior_probability_threshold: (a float)
    requires: prior_weighting
prior_weighting: (a float)
save_posteriors: (a boolean)
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
use_mixture_model_proportions: (a boolean)
    requires: posterior_formulation
use_random_seed: (a boolean, nipy default value: True)
    use random seed value over constant
    flag: --use-random-seed %d

```

Outputs:

```

classified_image: (an existing file name)
posteriors: (a list of items which are a file name)

```

BrainExtraction[Link to code](#)Wraps command **antsBrainExtraction.sh****Examples**

```

>>> from nipy.interfaces.ants.segmentation import BrainExtraction
>>> brainextraction = BrainExtraction()
>>> brainextraction.inputs.dimension = 3
>>> brainextraction.inputs.anatomical_image = 'T1.nii.gz'
>>> brainextraction.inputs.brain_template = 'study_template.nii.gz'
>>> brainextraction.inputs.brain_probability_mask =
↳ 'ProbabilityMaskOfStudyTemplate.nii.gz'
>>> brainextraction.cmdline
'antsBrainExtraction.sh -a T1.nii.gz -m ProbabilityMaskOfStudyTemplate.nii.gz -e_
↳ study_template.nii.gz -d 3 -s nii.gz -o highres001_'

```

Inputs:

```

[Mandatory]
anatomical_image: (an existing file name)
    Structural image, typically T1. If more than one anatomical image is
    specified, subsequently specified images are used during the
    segmentation process. However, only the first image is used in the
    registration of priors. Our suggestion would be to specify the T1 as

```

```

        the first image. Anatomical template created using e.g. LPBA40 data
        set with buildtemplateparallel.sh in ANTs.
        flag: -a %s
brain_probability_mask: (an existing file name)
        Brain probability mask created using e.g. LPBA40 data set which have
        brain masks defined, and warped to anatomical template and averaged
        resulting in a probability image.
        flag: -m %s
brain_template: (an existing file name)
        Anatomical template created using e.g. LPBA40 data set
        with buildtemplateparallel.sh in ANTs.
        flag: -e %s

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
debug: (a boolean)
        If > 0, runs a faster version of the script. Only for testing.
        Implies -u 0. Requires single thread computation for complete
        reproducibility.
        flag: -z 1
dimension: (3 or 2, nipy default value: 3)
        image dimension (2 or 3)
        flag: -d %d
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
extraction_registration_mask: (an existing file name)
        Mask (defined in the template space) used during registration for
        brain extraction. To limit the metric computation to a specific
        region.
        flag: -f %s
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
image_suffix: (a unicode string, nipy default value: nii.gz)
        any of standard ITK formats, nii.gz is default
        flag: -s %s
keep_temporary_files: (an integer (int or long))
        Keep brain extraction/segmentation warps, etc (default = 0).
        flag: -k %d
num_threads: (an integer (int or long), nipy default value: 1)
        Number of ITK threads to use
out_prefix: (a unicode string, nipy default value: highres001_)
        Prefix that is prepended to all output files (default =
        highres001_)
        flag: -o %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
use_floatingpoint_precision: (0 or 1)
        Use floating point precision in registrations (default = 0)
        flag: -q %d
use_random_seeding: (0 or 1)
        Use random number generated from system clock in Atropos (default =

```



```
1)
flag: -u %d
```

Outputs:

```
BrainExtractionBrain: (an existing file name)
    brain extraction image
BrainExtractionCSF: (an existing file name)
    segmentation mask with only CSF
BrainExtractionGM: (an existing file name)
    segmentation mask with only grey matter
BrainExtractionInitialAffine: (an existing file name)
BrainExtractionInitialAffineFixed: (an existing file name)
BrainExtractionInitialAffineMoving: (an existing file name)
BrainExtractionLaplacian: (an existing file name)
BrainExtractionMask: (an existing file name)
    brain extraction mask
BrainExtractionPrior0GenericAffine: (an existing file name)
BrainExtractionPrior1InverseWarp: (an existing file name)
BrainExtractionPrior1Warp: (an existing file name)
BrainExtractionPriorWarped: (an existing file name)
BrainExtractionSegmentation: (an existing file name)
    segmentation mask with CSF, GM, and WM
BrainExtractionTemplateLaplacian: (an existing file name)
BrainExtractionTmp: (an existing file name)
BrainExtractionWM: (an existing file name)
    segmenration mask with only white matter
N4Corrected0: (an existing file name)
    N4 bias field corrected image
N4Truncated0: (an existing file name)
```

CorticalThickness[Link to code](#)Wraps command **antsCorticalThickness.sh****Examples**

```
>>> from nipy.interfaces.ants.segmentation import CorticalThickness
>>> corticalthickness = CorticalThickness()
>>> corticalthickness.inputs.dimension = 3
>>> corticalthickness.inputs.anatomical_image = 'T1.nii.gz'
>>> corticalthickness.inputs.brain_template = 'study_template.nii.gz'
>>> corticalthickness.inputs.brain_probability_mask =
↳ 'ProbabilityMaskOfStudyTemplate.nii.gz'
>>> corticalthickness.inputs.segmentation_priors = ['BrainSegmentationPrior01.nii.
↳ gz',
...
↳ gz',
...
↳ gz',
...
↳ gz']
>>> corticalthickness.inputs.t1_registration_template = 'brain_study_template.nii.
↳ gz'
>>> corticalthickness.cmdline
'antsCorticalThickness.sh -a T1.nii.gz -m ProbabilityMaskOfStudyTemplate.nii.gz -
↳ e study_template.nii.gz -d 3 -s nii.gz -o antsCT_ -p nipy_priors/
↳ BrainSegmentationPrior%02d.nii.gz -t brain_study_template.nii.gz'
```

Inputs:

```

[Mandatory]
anatomical_image: (an existing file name)
    Structural *intensity* image, typically T1.If more than one
    anatomical image is specified,subsequently specified images are used
    during thesegmentation process. However, only the firstimage is used
    in the registration of priors.Our suggestion would be to specify the
    T1as the first image.
    flag: -a %s
brain_probability_mask: (an existing file name)
    brain probability mask in template space
    flag: -m %s
brain_template: (an existing file name)
    Anatomical *intensity* template (possibly created using apopulation
    data set with buildtemplateparallel.sh in ANTs).This template is
    *not* skull-stripped.
    flag: -e %s
segmentation_priors: (a list of items which are an existing file
    name)
    flag: -p %s
t1_registration_template: (an existing file name)
    Anatomical *intensity* template(assumed to be skull-stripped). A
    commoncase would be where this would be the sametemplate as
    specified in the -e option whichis not skull stripped.
    flag: -t %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
b_spline_smoothing: (a boolean)
    Use B-spline SyN for registrations and B-splineexponential mapping
    in DiReCT.
    flag: -v
cortical_label_image: (an existing file name)
    Cortical ROI labels to use as a prior for ATITH.
debug: (a boolean)
    If > 0, runs a faster version of the script.Only for testing.
    Implies -u 0.Requires single thread computation for complete
    reproducibility.
    flag: -z 1
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    flag: -d %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
extraction_registration_mask: (an existing file name)
    Mask (defined in the template space) used during registration for
    brain extraction.
    flag: -f %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
image_suffix: (a unicode string, nipy default value: nii.gz)

```

```

    any of standard ITK formats, nii.gz is default
    flag: -s %s
keep_temporary_files: (an integer (int or long))
    Keep brain extraction/segmentation warps, etc (default = 0).
    flag: -k %d
label_propagation: (a unicode string)
    Incorporate a distance prior one the posterior formulation. Should
    be of the form 'label[lambda,boundaryProbability]' where label is a
    value of 1,2,3,... denoting label ID. The label probability for
    anything outside the current label = boundaryProbability * exp(
    -lambda * distanceFromBoundary ) Intuitively, smaller lambda values
    will increase the spatial capture range of the distance prior. To
    apply to all label values, simply omit specifying the label, i.e. -l
    [lambda,boundaryProbability].
    flag: -l %s
max_iterations: (an integer (int or long))
    ANTS registration max iterations (default = 100x100x70x20)
    flag: -i %d
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_prefix: (a unicode string, nipy default value: antsCT_)
    Prefix that is prepended to all output files (default = antsCT_)
    flag: -o %s
posterior_formulation: (a unicode string)
    Atropos posterior formulation and whether or not to use mixture model
    proportions. e.g. 'Socrates[1]' (default) or 'Aristotle[1]'. Choose the
    latter if you want use the distance priors (see also the -l option for
    label propagation control).
    flag: -b %s
prior_segmentation_weight: (a float)
    Atropos spatial prior *probability* weight for the segmentation
    flag: -w %f
quick_registration: (a boolean)
    If = 1, use antsRegistrationSyNQuick.sh as the basis for
    registration during brain extraction, brain segmentation,
    and (optional) normalization to a template. Otherwise use
    antsRegistrationSyN.sh (default = 0).
    flag: -q 1
segmentation_iterations: (an integer (int or long))
    N4 -> Atropos -> N4 iterations during segmentation (default = 3)
    flag: -n %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
use_floatingpoint_precision: (0 or 1)
    Use floating point precision in registrations (default = 0)
    flag: -j %d
use_random_seeding: (0 or 1)
    Use random number generated from system clock in Atropos (default =
    1)
    flag: -u %d

```

Outputs:

```

BrainExtractionMask: (an existing file name)
    brain extraction mask
BrainSegmentation: (an existing file name)

```

```

    brain segmentaion image
BrainSegmentationN4: (an existing file name)
    N4 corrected image
BrainSegmentationPosteriors: (a list of items which are an existing
    file name)
    Posterior probability images
BrainVolumes: (an existing file name)
    Brain volumes as text
CorticalThickness: (an existing file name)
    cortical thickness file
CorticalThicknessNormedToTemplate: (an existing file name)
    Normalized cortical thickness
SubjectToTemplate0GenericAffine: (an existing file name)
    Template to subject inverse affine
SubjectToTemplate1Warp: (an existing file name)
    Template to subject inverse warp
SubjectToTemplateLogJacobian: (an existing file name)
    Template to subject log jacobian
TemplateToSubject0Warp: (an existing file name)
    Template to subject warp
TemplateToSubject1GenericAffine: (an existing file name)
    Template to subject affine

```

DenoiseImage

[Link to code](#)

Wraps command **DenoiseImage**

Examples

```

>>> import copy
>>> from nipy.interfaces.ants import DenoiseImage
>>> denoise = DenoiseImage()
>>> denoise.inputs.dimension = 3
>>> denoise.inputs.input_image = 'im1.nii'
>>> denoise.cmdline
'DenoiseImage -d 3 -i im1.nii -n Gaussian -o im1_noise_corrected.nii -s 1'

```

```

>>> denoise_2 = copy.deepcopy(denoise)
>>> denoise_2.inputs.output_image = 'output_corrected_image.nii.gz'
>>> denoise_2.inputs.noise_model = 'Rician'
>>> denoise_2.inputs.shrink_factor = 2
>>> denoise_2.cmdline
'DenoiseImage -d 3 -i im1.nii -n Rician -o output_corrected_image.nii.gz -s 2'

```

```

>>> denoise_3 = DenoiseImage()
>>> denoise_3.inputs.input_image = 'im1.nii'
>>> denoise_3.inputs.save_noise = True
>>> denoise_3.cmdline
'DenoiseImage -i im1.nii -n Gaussian -o [ im1_noise_corrected.nii, im1_noise.nii_
↵] -s 1'

```

Inputs:

```

[Mandatory]
input_image: (an existing file name)

```

```

    A scalar image is expected as input for noise correction.
    flag: -i %s
save_noise: (a boolean, nipy default value: False)
    True if the estimated noise should be saved to file.
    mutually_exclusive: noise_image

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dimension: (2 or 3 or 4)
    This option forces the image to be treated as a specified-
    dimensional image. If not specified, the program tries to infer the
    dimensionality from the input image.
    flag: -d %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
noise_image: (a file name)
    Filename for the estimated noise.
noise_model: ('Gaussian' or 'Rician', nipy default value: Gaussian)
    Employ a Rician or Gaussian noise model.
    flag: -n %s
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
output_image: (a file name)
    The output consists of the noise corrected version of the input
    image.
    flag: -o %s
shrink_factor: (an integer (int or long), nipy default value: 1)
    Running noise correction on large images can be time consuming. To
    lessen computation time, the input image can be resampled. The
    shrink factor, specified as a single integer, describes this
    resampling. Shrink factor = 1 is the default.
    flag: -s %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Verbose output.
    flag: -v

```

Outputs:

```

noise_image: (a file name)
output_image: (an existing file name)

```

JointFusion[Link to code](#)Wraps command **jointfusion**

Examples

```
>>> from nipyre.interfaces.ants import JointFusion
>>> at = JointFusion()
>>> at.inputs.dimension = 3
>>> at.inputs.modalities = 1
>>> at.inputs.method = 'Joint[0.1,2]'
>>> at.inputs.output_label_image = 'fusion_labelimage_output.nii'
>>> at.inputs.warped_intensity_images = ['im1.nii',
...                                     'im2.nii',
...                                     'im3.nii']
>>> at.inputs.warped_label_images = ['segmentation0.nii.gz',
...                                  'segmentation1.nii.gz',
...                                  'segmentation1.nii.gz']
>>> at.inputs.target_image = 'T1.nii'
>>> at.cmdline
'jointfusion 3 1 -m Joint[0.1,2] -tg T1.nii -g im1.nii -g im2.nii -g im3.nii -l_
↪segmentation0.nii.gz -l segmentation1.nii.gz -l segmentation1.nii.gz fusion_
↪labelimage_output.nii'
```

```
>>> at.inputs.method = 'Joint'
>>> at.inputs.alpha = 0.5
>>> at.inputs.beta = 1
>>> at.inputs.patch_radius = [3,2,1]
>>> at.inputs.search_radius = [1,2,3]
>>> at.cmdline
'jointfusion 3 1 -m Joint[0.5,1] -rp 3x2x1 -rs 1x2x3 -tg T1.nii -g im1.nii -g im2.
↪nii -g im3.nii -l segmentation0.nii.gz -l segmentation1.nii.gz -l segmentation1.
↪nii.gz fusion_labelimage_output.nii'
```

Inputs:

```
[Mandatory]
dimension: (3 or 2 or 4, nipyre default value: 3)
    image dimension (2, 3, or 4)
    flag: %d, position: 0
modalities: (an integer (int or long))
    Number of modalities or features
    flag: %d, position: 1
output_label_image: (a file name)
    Output fusion label map image
    flag: %s, position: -1
target_image: (a list of items which are an existing file name)
    Target image(s)
    flag: -tg %s...
warped_intensity_images: (a list of items which are an existing file
    name)
    Warped atlas images
    flag: -g %s...
warped_label_images: (a list of items which are an existing file
    name)
    Warped atlas segmentations
    flag: -l %s...

[Optional]
alpha: (a float, nipyre default value: 0.0)
    Regularization term added to matrix Mx for inverse
    requires: method
```

```

args: (a unicode string)
    Additional parameters to the command
    flag: %s
atlas_group_id: (a list of items which are a value of class 'int')
    Assign a group ID for each atlas
    flag: -gp %d...
atlas_group_weights: (a list of items which are a value of class
    'int')
    Assign the voting weights to each atlas group
    flag: -gpw %d...
beta: (an integer (int or long), nipyte default value: 0)
    Exponent for mapping intensity difference to joint error
    requires: method
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
exclusion_region: (an existing file name)
    Specify an exclusion region for the given label.
    flag: -x %s
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
method: (a unicode string, nipyte default value: )
    Select voting method. Options: Joint (Joint Label Fusion). May be
    followed by optional parameters in brackets, e.g., -m Joint[0.1,2]
    flag: -m %s
num_threads: (an integer (int or long), nipyte default value: 1)
    Number of ITK threads to use
patch_radius: (a list of items which are a value of class 'int')
    Patch radius for similarity measures, scalar or vector. Default:
    2x2x2
    flag: -rp %s
search_radius: (a list of items which are a value of class 'int')
    Local search radius. Default: 3x3x3
    flag: -rs %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```
output_label_image: (an existing file name)
```

KellyKapowski

[Link to code](#)

Wraps command **KellyKapowski**

Nipyte Interface to ANTs' KellyKapowski, also known as DiReCT.

DiReCT is a registration based estimate of cortical thickness. It was published in S. R. Das, B. B. Avants, M. Grossman, and J. C. Gee, Registration based cortical thickness measurement, Neuroimage 2009, 45:867–879.

Examples

```

>>> from nipyte.interfaces.ants.segmentation import KellyKapowski
>>> kk = KellyKapowski()

```

```

>>> kk.inputs.dimension = 3
>>> kk.inputs.segmentation_image = "segmentation0.nii.gz"
>>> kk.inputs.convergence = "[45,0.0,10]"
>>> kk.inputs.gradient_step = 0.025
>>> kk.inputs.smoothing_variance = 1.0
>>> kk.inputs.smoothing_velocity_field = 1.5
>>> #kk.inputs.use_bspline_smoothing = False
>>> kk.inputs.number_integration_points = 10
>>> kk.inputs.thickness_prior_estimate = 10
>>> kk.cmdline
u'KellyKapowski --convergence "[45,0.0,10]" --output "[segmentation0_cortical_
↳thickness.nii.gz,segmentation0_warped_white_matter.nii.gz]" --image-
↳dimensionality 3 --gradient-step 0.025000 --number-of-integration-points 10 --
↳segmentation-image "[segmentation0.nii.gz,2,3]" --smoothing-variance 1.000000 --
↳smoothing-velocity-field-parameter 1.500000 --thickness-prior-estimate 10.000000
↳'

```

Inputs:

```

[Mandatory]
segmentation_image: (an existing file name)
    A segmentation image must be supplied labeling the gray and white
    matters.
    Default values = 2 and 3, respectively.
    flag: --segmentation-image "%s"

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
convergence: (a unicode string, nipy default value: )
    Convergence is determined by fitting a line to the normalized energy
    profile of
    the last N iterations (where N is specified by the window size) and
    determining
    the slope which is then compared with the convergence threshold.
    flag: --convergence "%s"
cortical_thickness: (a file name)
    Filename for the cortical thickness.
    flag: --output "%s"
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    flag: --image-dimensionality %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gradient_step: (a float, nipy default value: 0.025)
    Gradient step size for the optimization.
    flag: --gradient-step %f
gray_matter_label: (an integer (int or long), nipy default value:
    2)
    The label value for the gray matter label in the segmentation_image.
gray_matter_prob_image: (an existing file name)
    In addition to the segmentation image, a gray matter probability
    image can be
    used. If no such image is supplied, one is created using the
    segmentation image

```



```

    and a variance of 1.0 mm.
    flag: --gray-matter-probability-image "%s"
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
max_invert_displacement_field_iters: (an integer (int or long))
    Maximum number of iterations for estimating the invert
    displacement field.
    flag: --maximum-number-of-invert-displacement-field-iterations %d
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
number_integration_points: (an integer (int or long))
    Number of compositions of the diffeomorphism per iteration.
    flag: --number-of-integration-points %d
smoothing_variance: (a float)
    Defines the Gaussian smoothing of the hit and total images.
    flag: --smoothing-variance %f
smoothing_velocity_field: (a float)
    Defines the Gaussian smoothing of the velocity field (default =
    1.5).
    If the b-spline smoothing option is chosen, then this defines the
    isotropic mesh spacing for the smoothing spline (default = 15).
    flag: --smoothing-velocity-field-parameter %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thickness_prior_estimate: (a float, nipy default value: 10)
    Provides a prior constraint on the final thickness measurement in
    mm.
    flag: --thickness-prior-estimate %f
thickness_prior_image: (an existing file name)
    An image containing spatially varying prior thickness values.
    flag: --thickness-prior-image "%s"
use_bspline_smoothing: (a boolean)
    Sets the option for B-spline smoothing of the velocity field.
    flag: --use-bspline-smoothing 1
warped_white_matter: (a file name)
    Filename for the warped white matter file.
white_matter_label: (an integer (int or long), nipy default value:
    3)
    The label value for the white matter label in the
    segmentation_image.
white_matter_prob_image: (an existing file name)
    In addition to the segmentation image, a white matter probability
    image can be
    used. If no such image is supplied, one is created using the
    segmentation image
    and a variance of 1.0 mm.
    flag: --white-matter-probability-image "%s"

```

Outputs:

```

cortical_thickness: (a file name)
    A thickness map defined in the segmented gray matter.
warped_white_matter: (a file name)
    A warped white matter image.

```

References:: None

LaplacianThickness

[Link to code](#)

Wraps command **LaplacianThickness**

Calculates the cortical thickness from an anatomical image

Examples

```
>>> from nipy.interfaces.ants import LaplacianThickness
>>> cort_thick = LaplacianThickness()
>>> cort_thick.inputs.input_wm = 'white_matter.nii.gz'
>>> cort_thick.inputs.input_gm = 'gray_matter.nii.gz'
>>> cort_thick.inputs.output_image = 'output_thickness.nii.gz'
>>> cort_thick.cmdline
'LaplacianThickness white_matter.nii.gz gray_matter.nii.gz output_thickness.nii.gz
↪ '
```

Inputs:

```
[Mandatory]
input_gm: (a file name)
    gray matter segmentation image
    flag: %s, position: 2
input_wm: (a file name)
    white matter segmentation image
    flag: %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dT: (a float)
    flag: dT=%d, position: 6
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
opt_tolerance: (a float)
    flag: optional-laplacian-tolerance=%d, position: 8
output_image: (a file name)
    name of output file
    flag: %s, position: 3
prior_thickness: (a float)
    flag: priorthickval=%d, position: 5
smooth_param: (a float)
    flag: smoothparam=%d, position: 4
sulcus_prior: (a boolean)
    flag: use-sulcus-prior, position: 7
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
output_image: (an existing file name)
               Cortical thickness
```

N4BiasFieldCorrection

[Link to code](#)

Wraps command **N4BiasFieldCorrection**

N4 is a variant of the popular N3 (nonparameteric nonuniform normalization) retrospective bias correction algorithm. Based on the assumption that the corruption of the low frequency bias field can be modeled as a convolution of the intensity histogram by a Gaussian, the basic algorithmic protocol is to iterate between deconvolving the intensity histogram by a Gaussian, remapping the intensities, and then spatially smoothing this result by a B-spline modeling of the bias field itself. The modifications from and improvements obtained over the original N3 algorithm are described in *[Tustison2010]*.

Examples

```
>>> import copy
>>> from nipy.interfaces.ants import N4BiasFieldCorrection
>>> n4 = N4BiasFieldCorrection()
>>> n4.inputs.dimension = 3
>>> n4.inputs.input_image = 'structural.nii'
>>> n4.inputs.bspline_fitting_distance = 300
>>> n4.inputs.shrink_factor = 3
>>> n4.inputs.n_iterations = [50,50,30,20]
>>> n4.cmdline
'N4BiasFieldCorrection --bspline-fitting [ 300 ] -d 3 --input-image structural.
↪nii --convergence [ 50x50x30x20 ] --output structural_corrected.nii --shrink-
↪factor 3'
```

```
>>> n4_2 = copy.deepcopy(n4)
>>> n4_2.inputs.convergence_threshold = 1e-6
>>> n4_2.cmdline
'N4BiasFieldCorrection --bspline-fitting [ 300 ] -d 3 --input-image structural.
↪nii --convergence [ 50x50x30x20, 1e-06 ] --output structural_corrected.nii --
↪shrink-factor 3'
```

```
>>> n4_3 = copy.deepcopy(n4_2)
>>> n4_3.inputs.bspline_order = 5
>>> n4_3.cmdline
'N4BiasFieldCorrection --bspline-fitting [ 300, 5 ] -d 3 --input-image structural.
↪nii --convergence [ 50x50x30x20, 1e-06 ] --output structural_corrected.nii --
↪shrink-factor 3'
```

```
>>> n4_4 = N4BiasFieldCorrection()
>>> n4_4.inputs.input_image = 'structural.nii'
>>> n4_4.inputs.save_bias = True
>>> n4_4.inputs.dimension = 3
>>> n4_4.cmdline
'N4BiasFieldCorrection -d 3 --input-image structural.nii --output [ structural_
↪corrected.nii, structural_bias.nii ]'
```

Inputs:

```
[Mandatory]
input_image: (a file name)
```

```

        image to apply transformation to (generally a coregistered
        functional)
        flag: --input-image %s
save_bias: (a boolean, nipy default value: False)
    True if the estimated bias should be saved to file.
    mutually_exclusive: bias_image

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bias_image: (a file name)
    Filename for the estimated bias.
bspline_fitting_distance: (a float)
    flag: --bspline-fitting %s
bspline_order: (an integer (int or long))
    requires: bspline_fitting_distance
convergence_threshold: (a float)
    requires: n_iterations
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    flag: -d %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_image: (a file name)
    flag: --mask-image %s
n_iterations: (a list of items which are an integer (int or long))
    flag: --convergence %s
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
output_image: (a unicode string)
    output file name
    flag: --output %s
shrink_factor: (an integer (int or long))
    flag: --shrink-factor %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
weight_image: (a file name)
    flag: --weight-image %s

```

Outputs:

```

bias_image: (an existing file name)
    Estimated bias
output_image: (an existing file name)
    Warped image

```

antsBrainExtraction[Link to code](#)Wraps command **antsBrainExtraction.sh**

Inputs:

```

[Mandatory]
anatomical_image: (an existing file name)
    Structural image, typically T1. If more than one anatomical image is
    specified, subsequently specified images are used during the
    segmentation process. However, only the first image is used in the
    registration of priors. Our suggestion would be to specify the T1 as
    the first image. Anatomical template created using e.g. LPBA40 data
    set with buildtemplateparallel.sh in ANTs.
    flag: -a %s
brain_probability_mask: (an existing file name)
    Brain probability mask created using e.g. LPBA40 data set which have
    brain masks defined, and warped to anatomical template and averaged
    resulting in a probability image.
    flag: -m %s
brain_template: (an existing file name)
    Anatomical template created using e.g. LPBA40 data set
    with buildtemplateparallel.sh in ANTs.
    flag: -e %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debug: (a boolean)
    If > 0, runs a faster version of the script. Only for testing.
    Implies -u 0. Requires single thread computation for complete
    reproducibility.
    flag: -z 1
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    flag: -d %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
extraction_registration_mask: (an existing file name)
    Mask (defined in the template space) used during registration for
    brain extraction. To limit the metric computation to a specific
    region.
    flag: -f %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
image_suffix: (a unicode string, nipy default value: nii.gz)
    any of standard ITK formats, nii.gz is default
    flag: -s %s
keep_temporary_files: (an integer (int or long))
    Keep brain extraction/segmentation warps, etc (default = 0).
    flag: -k %d
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_prefix: (a unicode string, nipy default value: highres001_)
    Prefix that is prepended to all output files (default =
    highres001_)
    flag: -o %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately

```

```

        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
    use_floatingpoint_precision: (0 or 1)
        Use floating point precision in registrations (default = 0)
    flag: -q %d
    use_random_seeding: (0 or 1)
        Use random number generated from system clock in Atropos (default =
        1)
    flag: -u %d

```

Outputs:

```

BrainExtractionBrain: (an existing file name)
    brain extraction image
BrainExtractionCSF: (an existing file name)
    segmentation mask with only CSF
BrainExtractionGM: (an existing file name)
    segmentation mask with only grey matter
BrainExtractionInitialAffine: (an existing file name)
BrainExtractionInitialAffineFixed: (an existing file name)
BrainExtractionInitialAffineMoving: (an existing file name)
BrainExtractionLaplacian: (an existing file name)
BrainExtractionMask: (an existing file name)
    brain extraction mask
BrainExtractionPrior0GenericAffine: (an existing file name)
BrainExtractionPrior1InverseWarp: (an existing file name)
BrainExtractionPrior1Warp: (an existing file name)
BrainExtractionPriorWarped: (an existing file name)
BrainExtractionSegmentation: (an existing file name)
    segmentation mask with CSF, GM, and WM
BrainExtractionTemplateLaplacian: (an existing file name)
BrainExtractionTmp: (an existing file name)
BrainExtractionWM: (an existing file name)
    segmenration mask with only white matter
N4Corrected0: (an existing file name)
    N4 bias field corrected image
N4Truncated0: (an existing file name)

```

antsCorticalThickness[Link to code](#)Wraps command **antsCorticalThickness.sh****Inputs:**

```

[Mandatory]
anatomical_image: (an existing file name)
    Structural *intensity* image, typically T1.If more than one
    anatomical image is specified,subsequently specified images are used
    during thesegmentation process. However, only the firstimage is used
    in the registration of priors.Our suggestion would be to specify the
    T1as the first image.
    flag: -a %s
brain_probability_mask: (an existing file name)
    brain probability mask in template space
    flag: -m %s
brain_template: (an existing file name)
    Anatomical *intensity* template (possibly created using apopulation
    data set with buildtemplateparallel.sh in ANTs).This template is

```

```

    *not* skull-stripped.
    flag: -e %s
segmentation_priors: (a list of items which are an existing file
    name)
    flag: -p %s
t1_registration_template: (an existing file name)
    Anatomical *intensity* template(assumed to be skull-stripped). A
    commoncase would be where this would be the sametemplate as
    specified in the -e option whichis not skull stripped.
    flag: -t %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
b_spline_smoothing: (a boolean)
    Use B-spline SyN for registrations and B-splineexponential mapping
    in DiReCT.
    flag: -v
cortical_label_image: (an existing file name)
    Cortical ROI labels to use as a prior for ATITH.
debug: (a boolean)
    If > 0, runs a faster version of the script.Only for testing.
    Implies -u 0.Requires single thread computation for complete
    reproducibility.
    flag: -z 1
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    flag: -d %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
extraction_registration_mask: (an existing file name)
    Mask (defined in the template space) used during registration for
    brain extraction.
    flag: -f %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
image_suffix: (a unicode string, nipy default value: nii.gz)
    any of standard ITK formats, nii.gz is default
    flag: -s %s
keep_temporary_files: (an integer (int or long))
    Keep brain extraction/segmentation warps, etc (default = 0).
    flag: -k %d
label_propagation: (a unicode string)
    Incorporate a distance prior one the posterior formulation. Should
    beof the form 'label[lambda,boundaryProbability]' where labelis a
    value of 1,2,3,... denoting label ID. The labelprobability for
    anything outside the current label = boundaryProbability * exp(
    -lambda * distanceFromBoundary )Intuitively, smaller lambda values
    will increase the spatial capturerrange of the distance prior. To
    apply to all label values, simply omitspecifying the label, i.e. -l
    [lambda,boundaryProbability].
    flag: -l %s
max_iterations: (an integer (int or long))
    ANTS registration max iterations(default = 100x100x70x20)

```

```

    flag: -i %d
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_prefix: (a unicode string, nipy default value: antsCT_)
    Prefix that is prepended to all output files (default = antsCT_)
    flag: -o %s
posterior_formulation: (a unicode string)
    Atropos posterior formulation and whether or notto use mixture model
    proportions.e.g 'Socrates[1]' (default) or 'Aristotle[1]'.Choose the
    latter if youwant use the distance priors (see also the -l optionfor
    label propagation control).
    flag: -b %s
prior_segmentation_weight: (a float)
    Atropos spatial prior *probability* weight forthe segmentation
    flag: -w %f
quick_registration: (a boolean)
    If = 1, use antsRegistrationSyNQuick.sh as the basis for
    registrationduring brain extraction, brain segmentation,
    and(optional) normalization to a template.Otherwise use
    antsRegistrationSyN.sh (default = 0).
    flag: -q 1
segmentation_iterations: (an integer (int or long))
    N4 -> Atropos -> N4 iterations during segmentation(default = 3)
    flag: -n %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
use_floatingpoint_precision: (0 or 1)
    Use floating point precision in registrations (default = 0)
    flag: -j %d
use_random_seeding: (0 or 1)
    Use random number generated from system clock in Atropos(default =
    1)
    flag: -u %d

```

Outputs:

```

BrainExtractionMask: (an existing file name)
    brain extraction mask
BrainSegmentation: (an existing file name)
    brain segmentaion image
BrainSegmentationN4: (an existing file name)
    N4 corrected image
BrainSegmentationPosteriors: (a list of items which are an existing
    file name)
    Posterior probability images
BrainVolumes: (an existing file name)
    Brain volumes as text
CorticalThickness: (an existing file name)
    cortical thickness file
CorticalThicknessNormedToTemplate: (an existing file name)
    Normalized cortical thickness
SubjectToTemplate0GenericAffine: (an existing file name)
    Template to subject inverse affine
SubjectToTemplate1Warp: (an existing file name)
    Template to subject inverse warp
SubjectToTemplateLogJacobian: (an existing file name)

```



```

    Template to subject log jacobian
TemplateToSubject0Warp: (an existing file name)
    Template to subject warp
TemplateToSubject1GenericAffine: (an existing file name)
    Template to subject affine

```

interfaces.ants.utils

AffineInitializer

[Link to code](#)

Wraps command **antsAffineInitializer**

Initialize an affine transform (as in antsBrainExtraction.sh)

```

>>> from nipy.interfaces.ants import AffineInitializer
>>> init = AffineInitializer()
>>> init.inputs.fixed_image = 'fixed1.nii'
>>> init.inputs.moving_image = 'moving1.nii'
>>> init.cmdline
'antsAffineInitializer 3 fixed1.nii moving1.nii transform.mat 15.000000 0.100000_
↪0 10'

```

Inputs:

```

[Mandatory]
fixed_image: (an existing file name)
    reference image
    flag: %s, position: 1
moving_image: (an existing file name)
    moving image
    flag: %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dimension: (3 or 2, nipy default value: 3)
    dimension
    flag: %s, position: 0
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
local_search: (an integer (int or long), nipy default value: 10)
    determines if a local optimization is run at each search point for
    the set number of iterations
    flag: %d, position: 7
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
out_file: (a file name, nipy default value: transform.mat)
    output transform file
    flag: %s, position: 3
principal_axes: (a boolean, nipy default value: False)
    whether the rotation is searched around an initial principal axis

```

```

        alignment.
        flag: %d, position: 6
radian_fraction: (0.0 <= a floating point number <= 1.0, nipy
        default value: 0.1)
        search this arc +/- principal axes
        flag: %f, position: 5
search_factor: (a float, nipy default value: 15.0)
        increments (degrees) for affine search
        flag: %f, position: 4
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
        output transform file

```

AverageAffineTransform

[Link to code](#)

Wraps command **AverageAffineTransform**

Examples

```

>>> from nipy.interfaces.ants import AverageAffineTransform
>>> avg = AverageAffineTransform()
>>> avg.inputs.dimension = 3
>>> avg.inputs.transforms = ['trans.mat', 'func_to_struct.mat']
>>> avg.inputs.output_affine_transform = 'MYtemplatewarp.mat'
>>> avg.cmdline
'AverageAffineTransform 3 MYtemplatewarp.mat trans.mat func_to_struct.mat'

```

Inputs:

```

[Mandatory]
dimension: (3 or 2)
        image dimension (2 or 3)
        flag: %d, position: 0
output_affine_transform: (a file name)
        Outputfname.txt: the name of the resulting transform.
        flag: %s, position: 1
transforms: (a list of items which are an existing file name)
        transforms to average
        flag: %s, position: 3

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the

```

```

        interface fails to run
num_threads: (an integer (int or long), nipy default value: 1)
        Number of ITK threads to use
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

affine_transform: (an existing file name)
        average transform file

```

AverageImages[Link to code](#)Wraps command **AverageImages****Examples**

```

>>> from nipy.interfaces.ants import AverageImages
>>> avg = AverageImages()
>>> avg.inputs.dimension = 3
>>> avg.inputs.output_average_image = "average.nii.gz"
>>> avg.inputs.normalize = True
>>> avg.inputs.images = ['rcls1.nii', 'rcls1.nii']
>>> avg.cmdline
'AverageImages 3 average.nii.gz 1 rcls1.nii rcls1.nii'

```

Inputs:

```

[Mandatory]
dimension: (3 or 2)
        image dimension (2 or 3)
        flag: %d, position: 0
images: (a list of items which are an existing file name)
        image to apply transformation to (generally a coregistered
        functional)
        flag: %s, position: 3
normalize: (a boolean)
        Normalize: if true, the 2nd image is divided by its mean. This will
        select the largest image to average into.
        flag: %d, position: 2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
num_threads: (an integer (int or long), nipy default value: 1)
        Number of ITK threads to use

```

```

output_average_image: (a file name, nipy default value:
    average.nii)
    the name of the resulting image.
    flag: %s, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

output_average_image: (an existing file name)
    average image file

```

CreateJacobianDeterminantImage[Link to code](#)Wraps command **CreateJacobianDeterminantImage****Examples**

```

>>> from nipy.interfaces.ants import CreateJacobianDeterminantImage
>>> jacobian = CreateJacobianDeterminantImage()
>>> jacobian.inputs.imageDimension = 3
>>> jacobian.inputs.deformationField = 'ants_Warp.nii.gz'
>>> jacobian.inputs.outputImage = 'out_name.nii.gz'
>>> jacobian.cmdline
'CreateJacobianDeterminantImage 3 ants_Warp.nii.gz out_name.nii.gz'

```

Inputs:

```

[Mandatory]
deformationField: (an existing file name)
    deformation transformation file
    flag: %s, position: 1
imageDimension: (3 or 2)
    image dimension (2 or 3)
    flag: %d, position: 0
outputImage: (a file name)
    output filename
    flag: %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
doLogJacobian: (0 or 1)
    return the log jacobian
    flag: %d, position: 3
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
num_threads: (an integer (int or long), nipy default value: 1)

```

```

    Number of ITK threads to use
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
useGeometric: (0 or 1)
    return the geometric jacobian
flag: %d, position: 4

```

Outputs:

```

jacobian_image: (an existing file name)
    jacobian image

```

MultiplyImages

[Link to code](#)Wraps command **MultiplyImages**

Examples

```

>>> from nipy.interfaces.ants import MultiplyImages
>>> test = MultiplyImages()
>>> test.inputs.dimension = 3
>>> test.inputs.first_input = 'moving2.nii'
>>> test.inputs.second_input = 0.25
>>> test.inputs.output_product_image = "out.nii"
>>> test.cmdline
'MultiplyImages 3 moving2.nii 0.25 out.nii'

```

Inputs:

```

[Mandatory]
dimension: (3 or 2)
    image dimension (2 or 3)
    flag: %d, position: 0
first_input: (an existing file name)
    image 1
    flag: %s, position: 1
output_product_image: (a file name)
    Outputfname.nii.gz: the name of the resulting image.
    flag: %s, position: 3
second_input: (an existing file name or a float)
    image 2 or multiplication weight
    flag: %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run

```

```
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
output_product_image: (an existing file name)
    average image file
```

interfaces.ants.visualization

ConvertScalarImageToRGB

[Link to code](#)Wraps command **ConvertScalarImageToRGB****Examples**

```
>>> from nipy.interfaces.ants.visualization import ConvertScalarImageToRGB
>>> converter = ConvertScalarImageToRGB()
>>> converter.inputs.dimension = 3
>>> converter.inputs.input_image = 'T1.nii.gz'
>>> converter.inputs.colormap = 'jet'
>>> converter.inputs.minimum_input = 0
>>> converter.inputs.maximum_input = 6
>>> converter.cmdline
'ConvertScalarImageToRGB 3 T1.nii.gz rgb.nii.gz none jet none 0 6 0 255'
```

Inputs:

```
[Mandatory]
colormap: (a unicode string, nipy default value: )
    Possible colormaps: grey, red, green, blue, copper, jet, hsv,
    spring, summer, autumn, winter, hot, cool, overunder, custom
    flag: %s, position: 4
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    flag: %d, position: 0
input_image: (an existing file name)
    Main input is a 3-D grayscale image.
    flag: %s, position: 1
maximum_input: (an integer (int or long))
    maximum input
    flag: %d, position: 7
minimum_input: (an integer (int or long))
    minimum input
    flag: %d, position: 6

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
custom_color_map_file: (a unicode string, nipy default value: none)
    custom color map file
```

```

    flag: %s, position: 5
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_image: (an existing file name, nipy default value: none)
    mask image
    flag: %s, position: 3
maximum_RGB_output: (an integer (int or long), nipy default value:
    255)
    flag: %d, position: 9
minimum_RGB_output: (an integer (int or long), nipy default value:
    0)
    flag: %d, position: 8
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
output_image: (a unicode string, nipy default value: rgb.nii.gz)
    rgb output image
    flag: %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

output_image: (an existing file name)
    converted RGB image

```

CreateTiledMosaic[Link to code](#)Wraps command **CreateTiledMosaic**

The program CreateTiledMosaic in conjunction with ConvertScalarImageToRGB provides useful functionality for common image analysis tasks. The basic usage of CreateTiledMosaic is to tile a 3-D image volume slice-wise into a 2-D image.

Examples

```

>>> from nipy.interfaces.ants.visualization import CreateTiledMosaic
>>> mosaic_slicer = CreateTiledMosaic()
>>> mosaic_slicer.inputs.input_image = 'T1.nii.gz'
>>> mosaic_slicer.inputs.rgb_image = 'rgb.nii.gz'
>>> mosaic_slicer.inputs.mask_image = 'mask.nii.gz'
>>> mosaic_slicer.inputs.output_image = 'output.png'
>>> mosaic_slicer.inputs.alpha_value = 0.5
>>> mosaic_slicer.inputs.direction = 2
>>> mosaic_slicer.inputs.pad_or_crop = '[-15x -50 , -15x -30 ,0]'
>>> mosaic_slicer.inputs.slices = '[2 ,100 ,160]'
>>> mosaic_slicer.cmdline
'CreateTiledMosaic -a 0.50 -d 2 -i T1.nii.gz -x mask.nii.gz -o output.png -p [ -
↪15x -50 , -15x -30 ,0] -r rgb.nii.gz -s [2 ,100 ,160]'

```

Inputs:

```

[Mandatory]
input_image: (an existing file name)
    Main input is a 3-D grayscale image.
    flag: -i %s
rgb_image: (an existing file name)
    An optional Rgb image can be added as an overlay. It must have the
    same image geometry as the input grayscale image.
    flag: -r %s

[Optional]
alpha_value: (a float)
    If an Rgb image is provided, render the overlay using the specified
    alpha parameter.
    flag: -a %.2f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
direction: (an integer (int or long))
    Specifies the direction of the slices. If no direction is specified,
    the direction with the coarsest spacing is chosen.
    flag: -d %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
flip_slice: (a unicode string)
    flipXxflipY
    flag: -f %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_image: (an existing file name)
    Specifies the ROI of the RGB voxels used.
    flag: -x %s
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
output_image: (a unicode string, nipy default value: output.png)
    The output consists of the tiled mosaic image.
    flag: -o %s
pad_or_crop: (a unicode string)
    argument passed to -p flag: [padVoxelWidth,<constantValue=0>][lowerPa
    dding[0]xlowerPadding[1],upperPadding[0]xupperPadding[1],constantVal
    ue]The user can specify whether to pad or crop a specified voxel-
    width boundary of each individual slice. For this program, cropping
    is simply padding with negative voxel-widths. If one pads (+), the
    user can also specify a constant pad value (default = 0). If a mask
    is specified, the user can use the mask to define the region, by
    using the keyword "mask" plus an offset, e.g. "-p mask+3".
    flag: -p %s
permute_axes: (a boolean)
    doPermute
    flag: -g
slices: (a unicode string)
    Number of slices to increment Slice1xSlice2xSlice3[numberOfSlicesToI
    ncrement,<minSlice=0>,<maxSlice=lastSlice>]
    flag: -s %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately

```



```
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
tile_geometry: (a unicode string)
    The tile geometry specifies the number of rows and columns in the
    output image. For example, if the user specifies "5x10", then 5 rows
    by 10 columns of slices are rendered. If R < 0 and C > 0 (or vice
    versa), the negative value is selected based on direction.
flag: -t %s
```

Outputs:

```
output_image: (an existing file name)
    image file
```


interfaces.brainsuite.brainsuite

BDP

[Link to code](#)

Wraps command **bdp.sh**

BrainSuite Diffusion Pipeline (BDP) enables fusion of diffusion and structural MRI information for advanced image and connectivity analysis. It provides various methods for distortion correction, co-registration, diffusion modeling (DTI and ODF) and basic ROI-wise statistic. BDP is a flexible and diverse tool which supports wide variety of diffusion datasets. For more information, please see:

<http://brainsuite.org/processing/diffusion/>

Examples

```
>>> from nipy.interfaces import brainsuite
>>> bdp = brainsuite.BDP()
>>> bdp.inputs.bfcFile = '/directory/subdir/prefix.bfc.nii.gz'
>>> bdp.inputs.inputDiffusionData = '/directory/subdir/prefix.dwi.nii.gz'
>>> bdp.inputs.BVecBValPair = ['/directory/subdir/prefix.dwi.bvec', '/directory/
↳subdir/prefix.dwi.bval']
>>> results = bdp.run()
```

Inputs:

```
[Mandatory]
BVecBValPair: (a list of from 2 to 2 items which are a value of class
    'str')
    Must input a list containing first the BVector file, then the BValue
    file (both must be absolute paths)
    Example: bdp.inputs.BVecBValPair =
    ['/directory/subdir/prefix.dwi.bvec',
    '/directory/subdir/prefix.dwi.bval'] The first item in the list
    specifies the filename of the file containing b-values for the
    diffusion scan. The b-value file must be a plain-text file and
    usually has an extension of .bval
    The second item in the list specifies the filename of the file
```

```

    containing the diffusion gradient directions (specified in the voxel
    coordinates of the input diffusion-weighted image)The b-vectors file
    must be a plain text file and usually has an extension of .bvec
    flag: --bvec %s --bval %s, position: -1
    mutually_exclusive: bMatrixFile
bMatrixFile: (a file name)
    Specifies the absolute path and filename of the file containing
    b-matrices for diffusion-weighted scans. The flag must be followed
    by the filename. This file must be a plain text file containing 3x3
    matrices for each diffusion encoding direction. It should contain
    zero matrices corresponding to b=0 images. This file usually has
    ".bmat" as its extension, and can be used to provide BDP with the
    more-accurate b-matrices as saved by some proprietary scanners. The
    b-matrices specified by the file must be in the voxel coordinates of
    the input diffusion weighted image (NifTI file). In case b-matrices
    are not known/calculated, bvec and .bval files can be used instead
    (see diffusionGradientFile and bValueFile).
    flag: --bmat %s, position: -1
    mutually_exclusive: BVecBValPair
bfcFile: (a file name)
    Specify absolute path to file produced by bfc. By default, bfc
    produces the file in the format: prefix.bfc.nii.gz
    flag: %s, position: 0
    mutually_exclusive: noStructuralRegistration
inputDiffusionData: (a file name)
    Specifies the absolute path and filename of the input diffusion data
    in 4D NIfTI-1 format. The flag must be followed by the filename.
    Only NIfTI-1 files with extension .nii or .nii.gz are supported.
    Furthermore, either bMatrixFile, or a combination of both bValueFile
    and diffusionGradientFile must be used to provide the necessary
    b-matrices/b-values and gradient vectors.
    flag: --nii %s, position: -2
noStructuralRegistration: (a boolean)
    Allows BDP to work without any structural input. This can useful
    when one is only interested in diffusion modelling part of BDP. With
    this flag only fieldmap-based distortion correction is supported.
    outPrefix can be used to specify fileprefix of the output filenames.
    Change dwiMask to define region of interest for diffusion modelling.
    flag: --no-structural-registration, position: 0
    mutually_exclusive: bfcFile

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bValRatioThreshold: (a float)
    Sets a threshold which is used to determine b=0 images. When there
    are no diffusion weighted image with b-value of zero, then BDP tries
    to use diffusion weighted images with a low b-value in place of b=0
    image. The diffusion images with minimum b-value is used as b=0
    image only if the ratio of the maximum and minimum b-value is more
    than the specified threshold. A lower value of threshold will allow
    diffusion images with higher b-value to be used as b=0 image. The
    default value of this threshold is set to 45, if this trait is not
    set.
    flag: --bval-ratio-threshold %f
customDiffusionLabel: (a file name)
    BDP supports custom ROIs in addition to those generated by

```

BrainSuite SVReg) for ROI-wise statistics calculation. The flag must be followed by the name of either a file (custom ROI file) or of a folder that contains one or more ROI files. All of the files must be in diffusion coordinate, i.e. the label files should overlay correctly with the diffusion scan in BrainSuite. These input label files are also transferred (and saved) to T1 coordinate for statistics in T1 coordinate. BDP uses nearest-neighborhood interpolation for this transformation. Only NIfTI files, with an extension of .nii or .nii.gz are supported. In order to avoid confusion with other ROI IDs in the statistic files, a 5-digit ROI ID is generated for each custom label found and the mapping of ID to label file is saved in the file fileprefix>.BDP_ROI_MAP.xml. Custom label files can also be generated by using the label painter tool in BrainSuite. See also customLabelXML

flag: --custom-diffusion-label %s

customLabelXML: (a file name)

BrainSuite saves a descriptions of the SVReg labels (ROI name, ID, color, and description) in an .xml file (brainsuite_labeldescription.xml). BDP uses the ROI ID"s from this xml file to report statistics. This flag allows for the use of a custom label description xml file. The flag must be followed by an xml filename. This can be useful when you want to limit the ROIs for which you compute statistics. You can also use custom xml files to name your own ROIs (assign ID"s) for custom labels. BrainSuite can save a label description in .xml format after using the label painter tool to create a ROI label. The xml file MUST be in the same format as BrainSuite"s label description file (see brainsuite_labeldescription.xml for an example). When this flag is used, NO 5-digit ROI ID is generated for custom label files and NO Statistics will be calculated for ROIs not identified in the custom xml file. See also customDiffusionLabel and customT1Label.

flag: --custom-label-xml %s

customT1Label: (a file name)

Same as customDiffusionLabel except that the label files specified must be in T1 coordinate, i.e. the label files should overlay correctly with the T1-weighted scan in BrainSuite. If the trait outputDiffusionCoordinates is also used then these input label files are also transferred (and saved) to diffusion coordinate for statistics in diffusion coordinate. BDP uses nearest-neighborhood interpolation for this transformation. See also customLabelXML.

flag: --custom-t1-label %s

dataSinkDelay: (a list of items which are a value of class 'str')

For use in parallel processing workflows including Brainsuite Cortical Surface Extraction sequence. Connect datasink out_file to dataSinkDelay to delay execution of BDP until dataSink has finished sinking outputs. In particular, BDP may be run after BFC has finished. For more information see <http://brainsuite.org/processing/diffusion/pipeline/>

flag: %s

dcorrRegMeasure: ('MI' or 'INVERSION-EPI' or 'INVERSION-T1' or 'INVERSION-BOTH' or 'BDP')

Defines the method for registration-based distortion correction. Possible methods are "MI", "INVERSION-EPI", "INVERSION-T1", "INVERSION-BOTH", and "BDP". MI method uses normalized mutual information based cost-function while estimating the distortion field. INVERSION-based method uses simpler cost function based on sum of squared difference by exploiting the known approximate contrast relationship in T1- and T2-weighted images. T2-weighted EPI

is inverted when INVERSION-EPI is used; T1-image is inverted when INVERSION-T1 is used; and both are inverted when INVERSION-BOTH is used. BDP method add the MI-based refinement after the correction using INVERSION-BOTH method. BDP is the default method when this trait is not set.

flag: `--dcorr-reg-method %s`

dcorrWeight: (a float)

Sets the (scalar) weighting parameter for regularization penalty in registration-based distortion correction. Set this trait to a single, non-negative number which specifies the weight. A large regularization weight encourages smoother distortion field at the cost of low measure of image similarity after distortion correction. On the other hand, a smaller regularization weight can result into higher measure of image similarity but with unrealistic and unsmooth distortion field. A weight of 0.5 would reduce the penalty to half of the default regularization penalty (By default, this weight is set to 1.0). Similarly, a weight of 2.0 would increase the penalty to twice of the default penalty.

flag: `--dcorr-regularization-wt %f`

dwiMask: (a file name)

Specifies the filename of the brain-mask file for diffusion data. This mask is used only for co-registration purposes and can affect overall quality of co-registration (see `t1Mask` for definition of brain mask for statistics computation). The mask must be a 3D volume and should be in the same coordinates as input Diffusion file/data (i.e. should overlay correctly with input diffusion data in BrainSuite). For best results, the mask should include only brain voxels (CSF voxels around brain is also acceptable). When this flag is not used, BDP will generate a pseudo mask using first `b=0` image volume and would save it as `fileprefix>.dwi.RSA.mask.nii.gz`. In case co-registration is not accurate with automatically generated pseudo mask, BDP should be re-run with a refined diffusion mask. The mask can be generated and/or edited in BrainSuite.

flag: `--dwi-mask %s`

echoSpacing: (a float)

Sets the echo spacing to `t` seconds, which is used for fieldmap-based distortion correction. This flag is required when using `fieldmapCorrection`

flag: `--echo-spacing=%f`

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipy default value: {})

Environment variables

estimateODF_3DShore: (a float)

Estimates ODFs using 3Dshore. Pass in diffusion time, in ms

flag: `--3dshore --diffusion_time_ms %f`

estimateODF_FRACT: (a boolean)

Estimates ODFs using the Funk-Radon and Cosine Transformation (FRACT). The outputs are saved in a separate directory with name "FRACT" and the ODFs can be visualized by loading the saved ".odf" file in BrainSuite.

flag: `--FRACT`

estimateODF_FRT: (a boolean)

Estimates ODFs using Funk-Radon Transformation (FRT). The coefficient maps for ODFs are saved in a separate directory with name "FRT" and the ODFs can be visualized by loading the saved ".odf" file in BrainSuite. The derived generalized-FA (GFA) maps are also saved in the output directory.

```

    flag: --FRT
estimateTensors: (a boolean)
    Estimates diffusion tensors using a weighted log-linear estimation
    and saves derived diffusion tensor parameters (FA, MD, axial,
    radial, L2, L3). This is the default behavior if no diffusion
    modeling flags are specified. The estimated diffusion tensors can be
    visualized by loading the saved *.eig.nii.gz file in BrainSuite. BDP
    reports diffusivity (MD, axial, radial, L2 and L3) in a unit which
    is reciprocal inverse of the unit of input b-value.
    flag: --tensors
fieldmapCorrection: (a file name)
    Use an acquired fieldmap for distortion correction. The fieldmap
    must have units of radians/second. Specify the filename of the
    fieldmap file. The field of view (FOV) of the fieldmap scan must
    cover the FOV of the diffusion scan. BDP will try to check the
    overlap of the FOV of the two scans and will issue a warning/error
    if the diffusion scan's FOV is not fully covered by the fieldmap's
    FOV. BDP uses all of the information saved in the NIfTI header to
    compute the FOV. If you get this error and think that it is
    incorrect, then it can be suppressed using the flag ignore-fieldmap-
    FOV. Neither the image matrix size nor the imaging grid resolution
    of the fieldmap needs to be the same as that of the diffusion scan,
    but the fieldmap must be pre-registered to the diffusion scan. BDP
    does NOT align the fieldmap to the diffusion scan, nor does it check
    the alignment of the fieldmap and diffusion scans. Only NIfTI files
    with extension of .nii or .nii.gz are supported. Fieldmap-based
    distortion correction also requires the echoSpacing. Also
    fieldmapCorrectionMethod allows you to define method for distortion
    correction. least squares is the default method.
    flag: --fieldmap-correction %s
    requires: echoSpacing
fieldmapCorrectionMethod: ('pixelshift' or 'leastsq')
    Defines the distortion correction method while using fieldmap.
    Possible methods are "pixelshift" and "leastsq". leastsq is the
    default method when this flag is not used. Pixel-shift (pixelshift)
    method uses image interpolation to un-distort the distorted
    diffusion images. Least squares (leastsq) method uses a physical
    model of distortion which is more accurate (and more computationally
    expensive) than pixel-shift method.
    flag: --fieldmap-correction-method %s
    mutually_exclusive: skipIntensityCorr
fieldmapSmooth: (a float)
    Applies 3D Gaussian smoothing with a standard deviation of S
    millimeters (mm) to the input fieldmap before applying distortion
    correction. This trait is only useful with fieldmapCorrection. Skip
    this trait for no smoothing.
    flag: --fieldmap-smooth3=%f
flagConfigFile: (a file name)
    Uses the defined file to specify BDP flags which can be useful for
    batch processing. A flag configuration file is a plain text file
    which can contain any number of BDP's optional flags (and their
    parameters) separated by whitespace. Everything coming after # until
    end-of-line is treated as comment and is ignored. If a flag is
    defined in configuration file and is also specified in the command
    used to run BDP, then the later get preference and overrides the
    definition in configuration file.
    flag: --flag-conf-file %s
forcePartialROIStats: (a boolean)

```

The field of view (FOV) of the diffusion and T1-weighted scans may differ significantly in some situations. This may result in partial acquisitions of some ROIs in the diffusion scan. By default, BDP does not compute statistics for partially acquired ROIs and shows warnings. This flag forces computation of statistics for all ROIs, including those which are partially acquired. When this flag is used, number of missing voxels are also reported for each ROI in statistics files. Number of missing voxels are reported in the same coordinate system as the statistics file.

flag: `--force-partial-roi-stats`

generateStats: (a boolean)

Generate ROI-wise statistics of estimated diffusion tensor parameters. Units of the reported statistics are same as that of the estimated tensor parameters (see `estimateTensors`). Mean, variance, and voxel counts of white matter (WM), grey matter (GM), and both WM and GM combined are written for each estimated parameter in a separate comma-separated value csv) file. BDP uses the ROI labels generated by Surface-Volume Registration (SVReg) in the BrainSuite extraction sequence. Specifically, it looks for labels saved in either `fileprefix>.svreg.corr.label.nii.gz` or `<fileprefix>.svreg.label.nii.gz`. In case both files are present, only the first file is used. Also see `customDiffusionLabel` and `customT1Label` for specifying your own ROIs. It is also possible to forgo computing the SVReg ROI-wise statistics and only compute stats with custom labels if SVReg label is missing. BDP also transfers (and saves) the label/mask files to appropriate coordinates before computing statistics. Also see `outputDiffusionCoordinates` for outputs in diffusion coordinate and `forcePartialROIStats` for an important note about field of view of diffusion and T1-weighted scans.

flag: `--generate-stats`

ignoreFieldmapFOV: (a boolean)

Supresses the error generated by an insufficient field of view of the input fieldmap and continues with the processing. It is useful only when used with fieldmap-based distortion correction. See `fieldmap-correction` for a detailed explanation.

flag: `--ignore-fieldmap-fov`

ignoreMemory: (a boolean)

Deactivates the inbuilt memory checks and forces BDP to run registration-based distortion correction at its default resolution even on machines with a low amount of memory. This may result in an out-of-memory error when BDP cannot allocate sufficient memory.

flag: `--ignore-memory`

ignore_exception: (a boolean, nipy default value: False)

Print an error message instead of throwing an exception in case the interface fails to run

lowMemory: (a boolean)

Activates low-memory mode. This will run the registration-based distortion correction at a lower resolution, which could result in a less-accurate correction. This should only be used when no other alternative is available.

flag: `--low-memory`

odfLambda: (a boolean)

Sets the regularization parameter, `lambda`, of the Laplace-Beltrami operator while estimating ODFs. The default value is set to 0.006 . This can be used to set the appropriate regularization for the input diffusion data.

flag: `--odf-lambda <L>`


```

onlyStats: (a boolean)
    Skip all of the processing (co-registration, distortion correction
    and tensor/ODF estimation) and directly start computation of
    statistics. This flag is useful when BDP was previously run on a
    subject (or fileprefix>) and statistics need to be (re-)computed
    later. This assumes that all the necessary files were generated
    earlier. All of the other flags MUST be used in the same way as they
    were in the initial BDP run that processed the data.
    flag: --generate-only-stats
outPrefix: (a unicode string)
    Specifies output fileprefix when noStructuralRegistration is used.
    The fileprefix can not start with a dash (-) and should be a simple
    string reflecting the absolute path to desired location, along with
    outPrefix. When this flag is not specified (and
    noStructuralRegistration is used) then the output files have same
    file-base as the input diffusion file. This trait is ignored when
    noStructuralRegistration is not used.
    flag: --output-fileprefix %s
outputDiffusionCoordinates: (a boolean)
    Enables estimation of diffusion tensors and/or ODFs (and statistics
    if applicable) in the native diffusion coordinate in addition to the
    default T1-coordinate. All native diffusion coordinate files are
    saved in a seperate folder named "diffusion_coord_outputs". In case
    statistics computation is required, it will also transform/save all
    label/mask files required to diffusion coordinate (see generateStats
    for details).
    flag: --output-diffusion-coordinate
outputSubdir: (a unicode string)
    By default, BDP writes out all the output (and intermediate) files
    in the same directory (or folder) as the BFC file. This flag allows
    to specify a sub-directory name in which output (and intermediate)
    files would be written. BDP will create the sub-directory in the
    same directory as BFC file. <directory_name> should be the name of
    the sub-directory without any path. This can be useful to organize
    all outputs generated by BDP in a separate sub-directory.
    flag: --output-subdir %s
phaseEncodingDirection: ('x' or 'x-' or 'y' or 'y-' or 'z' or 'z-')
    Specifies the phase-encoding direction of the EPI (diffusion)
    images. It is same as the dominant direction of distortion in the
    images. This information is used to constrain the distortion
    correction along the specified direction. Directions are represented
    by any one of x, x-, y, y-, z or z-. "x" direction increases towards
    the right side of the subject, while "x-" increases towards the left
    side of the subject. Similarly, "y" and "y-" are along the anterior-
    posterior direction of the subject, and "z" & "z-" are along the
    inferior-superior direction. When this flag is not used, BDP uses
    "y" as the default phase-encoding direction.
    flag: --dir=%s
rigidRegMeasure: ('MI' or 'INVERSION' or 'BDP')
    Defines the similarity measure to be used for rigid registration.
    Possible measures are "MI", "INVERSION" and "BDP". MI measure uses
    normalized mutual information based cost function. INVERSION measure
    uses simpler cost function based on sum of squared difference by
    exploiting the approximate inverse-contrast relationship in T1- and
    T2-weighted images. BDP measure combines MI and INVERSION. It starts
    with INVERSION measure and refines the result with MI measure. BDP
    is the default measure when this trait is not set.
    flag: --rigid-reg-measure %s

```

```

skipDistortionCorr: (a boolean)
    Skips distortion correction completely and performs only a rigid
    registration of diffusion and T1-weighted image. This can be useful
    when the input diffusion images do not have any distortion or they
    have been corrected for distortion.
    flag: --no-distortion-correction
skipIntensityCorr: (a boolean)
    Disables intensity correction when performing distortion correction.
    Intensity correction can change the noise distribution in the
    corrected image, but it does not affect estimated diffusion
    parameters like FA, etc.
    flag: --no-intensity-correction
    mutually_exclusive: fieldmapCorrectionMethod
skipNonuniformityCorr: (a boolean)
    Skips intensity non-uniformity correction in b=0 image for
    registration-based distortion correction. The intensity non-
    uniformity correction does not affect any diffusion modeling.
    flag: --no-nonuniformity-correction
t1Mask: (a file name)
    Specifies the filename of the brain-mask file for input T1-weighted
    image. This mask can be same as the brain mask generated during
    BrainSuite extraction sequence. For best results, the mask should
    not include any extra-meningial tissues from T1-weighted image. The
    mask must be in the same coordinates as input T1-weighted image
    (i.e. should overlay correctly with input <fileprefix>.bfc.nii.gz
    file in BrainSuite). This mask is used for co-registration and
    defining brain boundary for statistics computation. The mask can be
    generated and/or edited in BrainSuite. In case
    outputDiffusionCoordinates is also used, this mask is first
    transformed to diffusion coordinate and the transformed mask is used
    for defining brain boundary in diffusion coordinates. When t1Mask is
    not set, BDP will try to use fileprefix>.mask.nii.gz as brain-mask.
    If <fileprefix>.mask.nii.gz is not found, then BDP will use the
    input <fileprefix>.bfc.nii.gz itself as mask (i.e. all non-zero
    voxels in <fileprefix>.bfc.nii.gz is assumed to constitute brain
    mask).
    flag: --t1-mask %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threads: (an integer (int or long))
    Sets the number of parallel process threads which can be used for
    computations to N, where N must be an integer. Default value of N is
    flag: --threads=%d
transformDataOnly: (a boolean)
    Skip all of the processing (co-registration, distortion correction
    and tensor/ODF estimation) and directly start transformation of
    defined custom volumes, mask and labels (using transformT1Volume,
    transformDiffusionVolume, transformT1Surface,
    transformDiffusionSurface, customDiffusionLabel, customT1Label).
    This flag is useful when BDP was previously run on a subject (or
    <fileprefix>) and some more data (volumes, mask or labels) need to
    be transformed across the T1-diffusion coordinate spaces. This
    assumes that all the necessary files were generated earlier and all
    of the other flags MUST be used in the same way as they were in the
    initial BDP run that processed the data.
    flag: --transform-data-only

```

```

transformDiffusionSurface: (a file name)
    Same as transformT1Volume, except that the .dfs files specified must
    be in diffusion coordinate, i.e. the surface files should overlay
    correctly with the diffusion scan in BrainSuite. The transformed
    files are written to the output directory with suffix ".T1_coord" in
    the filename. See also transformT1Volume.
    flag: --transform-diffusion-surface %s
transformDiffusionVolume: (a file name)
    This flag allows to define custom volumes in diffusion coordinate
    which would be transformed into T1 coordinate in a rigid fashion.
    The flag must be followed by the name of either a NIfTI file or of a
    folder that contains one or more NIfTI files. All of the files must
    be in diffusion coordinate, i.e. the files should overlay correctly
    with the diffusion scan in BrainSuite. Only NIfTI files with an
    extension of .nii or .nii.gz are supported. The transformed files
    are written to the output directory with suffix ".T1_coord" in the
    filename and will not be corrected for distortion, if any. The trait
    transformInterpolation can be used to define the type of
    interpolation that would be used (default is set to linear). If you
    are attempting to transform a label file or mask file, use "nearest"
    interpolation method with transformInterpolation. See also
    transformT1Volume and transformInterpolation
    flag: --transform-diffusion-volume %s
transformInterpolation: ('linear' or 'nearest' or 'cubic' or
    'spline')
    Defines the type of interpolation method which would be used while
    transforming volumes defined by transformT1Volume and
    transformDiffusionVolume. Possible methods are "linear", "nearest",
    "cubic" and "spline". By default, "linear" interpolation is used.
    flag: --transform-interpolation %s
transformT1Surface: (a file name)
    Similar to transformT1Volume, except that this flag allows
    transforming surfaces (instead of volumes) in T1 coordinate into
    diffusion coordinate in a rigid fashion. The flag must be followed
    by the name of either a .dfs file or of a folder that contains one
    or more dfs files. All of the files must be in T1 coordinate, i.e.
    the files should overlay correctly with the T1-weighted scan in
    BrainSuite. The transformed files are written to the output
    directory with suffix "D_coord" in the filename.
    flag: --transform-t1-surface %s
transformT1Volume: (a file name)
    Same as transformDiffusionVolume except that files specified must be
    in T1 coordinate, i.e. the files should overlay correctly with the
    input <fileprefix>.bfc.nii.gz files in BrainSuite. BDP transforms
    these data/images from T1 coordinate to diffusion coordinate. The
    transformed files are written to the output directory with suffix
    ".D_coord" in the filename. See also transformDiffusionVolume and
    transformInterpolation.
    flag: --transform-t1-volume %s

```

Outputs:

None

Bfc

[Link to code](#)

Wraps command **bfc**

bias field corrector (BFC) This program corrects gain variation in T1-weighted MRI.
<http://brainsuite.org/processing/surfaceextraction/bfc/>

Examples

```
>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> bfc = brainsuite.Bfc()
>>> bfc.inputs.inputMRIFile = example_data('structural.nii')
>>> bfc.inputs.inputMaskFile = example_data('mask.nii')
>>> results = bfc.run()
```

Inputs:

```
[Mandatory]
inputMRIFile: (a file name)
    input skull-stripped MRI volume
    flag: -i %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
biasEstimateConvergenceThreshold: (a float)
    bias estimate convergence threshold (values > 0.1 disable)
    flag: --beps %f
biasEstimateSpacing: (an integer (int or long))
    bias sample spacing (voxels)
    flag: -s %d
biasFieldEstimatesOutputPrefix: (a unicode string)
    save iterative bias field estimates as <prefix>.n.field.nii.gz
    flag: --biasprefix %s
biasRange: ('low' or 'medium' or 'high')
    Preset options for bias_model
    low: small bias model [0.95,1.05]
    medium: medium bias model [0.90,1.10]
    high: high bias model [0.80,1.20]
    flag: %s
controlPointSpacing: (an integer (int or long))
    control point spacing (voxels)
    flag: -c %d
convergenceThreshold: (a float)
    convergence threshold
    flag: --eps %f
correctWholeVolume: (a boolean)
    apply correction field to entire volume
    flag: --extrapolate
correctedImagesOutputPrefix: (a unicode string)
    save iterative corrected images as <prefix>.n.bfc.nii.gz
    flag: --prefix %s
correctionScheduleFile: (a file name)
    list of parameters
    flag: --schedule %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
histogramRadius: (an integer (int or long))
```

```

        histogram radius (voxels)
        flag: -r %d
    histogramType: ('ellipse' or 'block')
        Options for type of histogram
        ellipse: use ellipsoid for ROI histogram
        block :use block for ROI histogram
        flag: %s
    ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
    inputMaskFile: (a file name)
        mask file
        flag: -m %s
    intermediate_file_type: ('analyze' or 'nifti' or 'gzippedAnalyze' or
        'gzippedNifti')
        Options for the format in which intermediate files are generated
        flag: %s
    iterativeMode: (a boolean)
        iterative mode (overrides -r, -s, -c, -w settings)
        flag: --iterate
    maxBias: (a float, nipy default value: 1.5)
        maximum allowed bias value
        flag: -U %f
    minBias: (a float, nipy default value: 0.5)
        minimum allowed bias value
        flag: -L %f
    outputBiasField: (a file name)
        save bias field estimate
        flag: --bias %s
    outputMRIVolume: (a file name)
        output bias-corrected MRI volume.If unspecified, output file name
        will be auto generated.
        flag: -o %s
    outputMaskedBiasField: (a file name)
        save bias field estimate (masked)
        flag: --maskedbias %s
    splineLambda: (a float)
        spline stiffness weighting parameter
        flag: -w %f
    terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
    timer: (a boolean)
        display timing information
        flag: --timer
    verbosityLevel: (an integer (int or long))
        verbosity level (0=silent)
        flag: -v %d

```

Outputs:

```

correctionScheduleFile: (a file name)
    path/name of schedule file
outputBiasField: (a file name)
    path/name of bias field output file
outputMRIVolume: (a file name)
    path/name of output file

```

```
outputMaskedBiasField: (a file name)
                        path/name of masked bias field output
```

Bse

[Link to code](#)

Wraps command **bse**

brain surface extractor (BSE) This program performs automated skull and scalp removal on T1-weighted MRI volumes.

<http://brainsuite.org/processing/surfaceextraction/bse/>

Examples

```
>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> bse = brainsuite.Bse()
>>> bse.inputs.inputMRIFile = example_data('structural.nii')
>>> results = bse.run()
```

Inputs:

```
[Mandatory]
inputMRIFile: (a file name)
               input MRI volume
               flag: -i %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
diffusionConstant: (a float, nipy default value: 25)
                   diffusion constant
                   flag: -d %f
diffusionIterations: (an integer (int or long), nipy default value:
                      3)
                     diffusion iterations
                     flag: -n %d
dilateFinalMask: (a boolean, nipy default value: True)
                  dilate final mask
                  flag: -p
edgeDetectionConstant: (a float, nipy default value: 0.64)
                       edge detection constant
                       flag: -s %f
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
noRotate: (a boolean)
           retain original orientation(default behavior will auto-rotate input
           NII files to LPI orientation)
           flag: --norotate
outputCortexFile: (a file name)
                  cortex file
                  flag: --cortex %s
```

```

outputDetailedBrainMask: (a file name)
    save detailed brain mask
    flag: --hires %s
outputDiffusionFilter: (a file name)
    diffusion filter output
    flag: --adf %s
outputEdgeMap: (a file name)
    edge map output
    flag: --edge %s
outputMRIVolume: (a file name)
    output brain-masked MRI volume. If unspecified, output file name
    will be auto generated.
    flag: -o %s
outputMaskFile: (a file name)
    save smooth brain mask. If unspecified, output file name will be
    auto generated.
    flag: --mask %s
radius: (a float, nipy default value: 1)
    radius of erosion/dilation filter
    flag: -r %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timer: (a boolean)
    show timing
    flag: --timer
trim: (a boolean, nipy default value: True)
    trim brainstem
    flag: --trim
verbosityLevel: (a float, nipy default value: 1)
    verbosity level (0=silent)
    flag: -v %f

```

Outputs:

```

outputCortexFile: (a file name)
    path/name of cortex file
outputDetailedBrainMask: (a file name)
    path/name of detailed brain mask
outputDiffusionFilter: (a file name)
    path/name of diffusion filter output
outputEdgeMap: (a file name)
    path/name of edge map output
outputMRIVolume: (a file name)
    path/name of brain-masked MRI volume
outputMaskFile: (a file name)
    path/name of smooth brain mask

```

Cerebro[Link to code](#)Wraps command **cerebro**

Cerebrum/cerebellum labeling tool This program performs automated labeling of cerebellum and cerebrum in T1 MRI. Input MRI should be skull-stripped or a brain-only mask should be provided.

<http://brainsuite.org/processing/surfaceextraction/cerebrum/>

Examples

```
>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> cerebro = brainsuite.Cerebro()
>>> cerebro.inputs.inputMRIFile = example_data('structural.nii')
>>> cerebro.inputs.inputAtlasMRIFile = 'atlasMRIVolume.img'
>>> cerebro.inputs.inputAtlasLabelFile = 'atlasLabels.img'
>>> cerebro.inputs.inputBrainMaskFile = example_data('mask.nii')
>>> results = cerebro.run()
```

Inputs:

```
[Mandatory]
inputAtlasLabelFile: (a file name)
    atlas labeling
    flag: --atlaslabels %s
inputAtlasMRIFile: (a file name)
    atlas MRI volume
    flag: --atlas %s
inputMRIFile: (a file name)
    input 3D MRI volume
    flag: -i %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
costFunction: (an integer (int or long), nipy default value: 2)
    0,1,2
    flag: -c %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputBrainMaskFile: (a file name)
    brain mask file
    flag: -m %s
keepTempFiles: (a boolean)
    don't remove temporary files
    flag: --keep
linearConvergence: (a float)
    linear convergence
    flag: --linconv %f
outputAffineTransformFile: (a file name)
    save affine transform to file.
    flag: --air %s
outputCerebrumMaskFile: (a file name)
    output cerebrum mask volume. If unspecified, output file name will
    be auto generated.
    flag: -o %s
outputLabelVolumeFile: (a file name)
    output labeled hemisphere/cerebrum volume. If unspecified, output
    file name will be auto generated.
    flag: -l %s
outputWarpTransformFile: (a file name)
```



```

        save warp transform to file.
        flag: --warp %s
tempDirectory: (a unicode string)
        specify directory to use for temporary files
        flag: --tempdir %s
tempDirectoryBase: (a unicode string)
        create a temporary directory within this directory
        flag: --tempdirbase %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
useCentroids: (a boolean)
        use centroids of data to initialize position
        flag: --centroids
verbosity: (an integer (int or long))
        verbosity level (0=silent)
        flag: -v %d
warpConvergence: (a float)
        warp convergence
        flag: --warpconv %f
warpLabel: (an integer (int or long))
        warp order (2,3,4,5,6,7,8)
        flag: --warplevel %d

```

Outputs:

```

outputAffineTransformFile: (a file name)
        path/name of affine transform file
outputCerebrumMaskFile: (a file name)
        path/name of cerebrum mask file
outputLabelVolumeFile: (a file name)
        path/name of label mask file
outputWarpTransformFile: (a file name)
        path/name of warp transform file

```

Cortex[Link to code](#)Wraps command **cortex**

cortex extractor This program produces a cortical mask using tissue fraction estimates and a co-registered cerebellum/hemisphere mask.

<http://brainsuite.org/processing/surfaceextraction/cortex/>

Examples

```

>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> cortex = brainsuite.Cortex()
>>> cortex.inputs.inputHemisphereLabelFile = example_data('mask.nii')
>>> cortex.inputs.inputTissueFractionFile = example_data('tissues.nii.gz')
>>> results = cortex.run()

```

Inputs:

```

[Mandatory]
inputHemisphereLabelFile: (a file name)

```

```

    hemisphere / lobe label volume
    flag: -h %s
inputTissueFractionFile: (a file name)
    tissue fraction file (32-bit float)
    flag: -f %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
computeGCBoundary: (a boolean)
    compute GM/CSF boundary
    flag: -g
computeWGBoundary: (a boolean, nipy default value: True)
    compute WM/GM boundary
    flag: -w
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
includeAllSubcorticalAreas: (a boolean, nipy default value: True)
    include all subcortical areas in WM mask
    flag: -a
outputCerebrumMask: (a file name)
    output structure mask. If unspecified, output file name will be auto
    generated.
    flag: -o %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timer: (a boolean)
    timing function
    flag: --timer
tissueFractionThreshold: (a float, nipy default value: 50.0)
    tissue fraction threshold (percentage)
    flag: -p %f
verbosity: (an integer (int or long))
    verbosity level
    flag: -v %d

```

Outputs:

```

outputCerebrumMask: (a file name)
    path/name of cerebrum mask

```

Dewisp[Link to code](#)**Wraps command dewisp**

dewisp removes wispy tendrils from cortex model binary masks. It does so based on graph theoretic analysis of connected components, similar to TCA. Each branch of the structure graph is analyzed to determine pinch points that indicate a likely error in segmentation that attaches noise to the image. The pinch threshold determines how many voxels the cross-section can be before it is considered part of the image.

<http://brainsuite.org/processing/surfaceextraction/dewisp/>

Examples

```
>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> dewisp = brainsuite.Dewisp()
>>> dewisp.inputs.inputMaskFile = example_data('mask.nii')
>>> results = dewisp.run()
```

Inputs:

```
[Mandatory]
inputMaskFile: (a file name)
    input file
    flag: -i %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
maximumIterations: (an integer (int or long))
    maximum number of iterations
    flag: -n %d
outputMaskFile: (a file name)
    output file. If unspecified, output file name will be auto
    generated.
    flag: -o %s
sizeThreshold: (an integer (int or long))
    size threshold
    flag: -t %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timer: (a boolean)
    time processing
    flag: --timer
verbosity: (an integer (int or long))
    verbosity
    flag: -v %d
```

Outputs:

```
outputMaskFile: (a file name)
    path/name of mask file
```

Dfs

[Link to code](#)

Wraps command **dfs**

Surface Generator Generates mesh surfaces using an isosurface algorithm.

<http://brainsuite.org/processing/surfaceextraction/inner-cortical-surface/>

Examples

```

>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> dfs = brainsuite.Dfs()
>>> dfs.inputs.inputVolumeFile = example_data('structural.nii')
>>> results = dfs.run()

```

Inputs:

```

[Mandatory]
inputVolumeFile: (a file name)
    input 3D volume
    flag: -i %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
curvatureWeighting: (a float, nipy default value: 5.0)
    curvature weighting
    flag: -w %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputShadingVolume: (a file name)
    shade surface model with data from image volume
    flag: -c %s
noNormalsFlag: (a boolean)
    do not compute vertex normals
    flag: --nonormals
nonZeroTessellation: (a boolean)
    tessellate non-zero voxels
    flag: -nz
    mutually_exclusive: nonZeroTessellation, specialTessellation
outputSurfaceFile: (a file name)
    output surface mesh file. If unspecified, output file name will be
    auto generated.
    flag: -o %s
postSmoothFlag: (a boolean)
    smooth vertices after coloring
    flag: --postsmooth
scalingPercentile: (a float)
    scaling percentile
    flag: -f %f
smoothingConstant: (a float, nipy default value: 0.5)
    smoothing constant
    flag: -a %f
smoothingIterations: (an integer (int or long), nipy default value:
    10)
    number of smoothing iterations
    flag: -n %d
specialTessellation: ('greater_than' or 'less_than' or 'equal_to')
    To avoid throwing a UserWarning, set tessellationThreshold first.
    Then set this attribute.

```

```

Usage: tessellate voxels greater_than, less_than, or equal_to
<tessellationThreshold>
flag: %s, position: -1
mutually_exclusive: nonZeroTessellation, specialTessellation
requires: tessellationThreshold
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
tessellationThreshold: (a float)
To be used with specialTessellation. Set this value first, then set
specialTessellation value.
Usage: tessellate voxels greater_than, less_than, or equal_to
<tessellationThreshold>
flag: %f
timer: (a boolean)
timing function
flag: --timer
verbosity: (an integer (int or long))
verbosity (0 = quiet)
flag: -v %d
zeroPadFlag: (a boolean)
zero-pad volume (avoids clipping at edges)
flag: -z

```

Outputs:

```

outputSurfaceFile: (a file name)
path/name of surface file

```

Hemisplit[Link to code](#)Wraps command **hemisplit**

Hemisphere splitter Splits a surface object into two separate surfaces given an input label volume. Each vertex is labeled left or right based on the labels being odd (left) or even (right). The largest contour on the split surface is then found and used as the separation between left and right.

Examples

```

>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> hemisplit = brainsuite.Hemisplit()
>>> hemisplit.inputs.inputSurfaceFile = 'input_surf.dfs'
>>> hemisplit.inputs.inputHemisphereLabelFile = 'label.nii'
>>> hemisplit.inputs.pialSurfaceFile = 'pial.dfs'
>>> results = hemisplit.run()

```

Inputs:

```

[Mandatory]
inputHemisphereLabelFile: (a file name)
input hemisphere label volume
flag: -l %s
inputSurfaceFile: (a file name)
input surface
flag: -i %s

```

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
outputLeftHemisphere: (a file name)
    output surface file, left hemisphere. If unspecified, output file
    name will be auto generated.
    flag: --left %s
outputLeftPialHemisphere: (a file name)
    output pial surface file, left hemisphere. If unspecified, output
    file name will be auto generated.
    flag: -pl %s
outputRightHemisphere: (a file name)
    output surface file, right hemisphere. If unspecified, output file
    name will be auto generated.
    flag: --right %s
outputRightPialHemisphere: (a file name)
    output pial surface file, right hemisphere. If unspecified, output
    file name will be auto generated.
    flag: -pr %s
pialSurfaceFile: (a file name)
    pial surface file -- must have same geometry as input surface
    flag: -p %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timer: (a boolean)
    timing function
    flag: --timer
verbosity: (an integer (int or long))
    verbosity (0 = silent)
    flag: -v %d

```

Outputs:

```

outputLeftHemisphere: (a file name)
    path/name of left hemisphere
outputLeftPialHemisphere: (a file name)
    path/name of left pial hemisphere
outputRightHemisphere: (a file name)
    path/name of right hemisphere
outputRightPialHemisphere: (a file name)
    path/name of right pial hemisphere

```

Pialmesh[Link to code](#)Wraps command **pialmesh**

pialmesh computes a pial surface model using an inner WM/GM mesh and a tissue fraction map.

<http://brainsuite.org/processing/surfaceextraction/pial/>

Examples

```
>>> from nipyre.interfaces import brainsuite
>>> from nipyre.testing import example_data
>>> pialmesh = brainsuite.Pialmesh()
>>> pialmesh.inputs.inputSurfaceFile = 'input_mesh.dfs'
>>> pialmesh.inputs.inputTissueFractionFile = 'frac_file.nii.gz'
>>> pialmesh.inputs.inputMaskFile = example_data('mask.nii')
>>> results = pialmesh.run()
```

Inputs:

```
[Mandatory]
inputMaskFile: (a file name)
    restrict growth to mask file region
    flag: -m %s
inputSurfaceFile: (a file name)
    input file
    flag: -i %s
inputTissueFractionFile: (a file name)
    floating point (32) tissue fraction image
    flag: -f %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
exportPrefix: (a unicode string)
    prefix for exporting surfaces if interval is set
    flag: --prefix %s
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
laplacianSmoothing: (a float, nipyre default value: 0.025)
    apply Laplacian smoothing
    flag: --smooth %f
maxThickness: (a float, nipyre default value: 20)
    maximum allowed tissue thickness
    flag: --max %f
normalSmoother: (a float, nipyre default value: 0.2)
    strength of normal smoother.
    flag: --nc %f
numIterations: (an integer (int or long), nipyre default value: 100)
    number of iterations
    flag: -n %d
outputInterval: (an integer (int or long), nipyre default value: 10)
    output interval
    flag: --interval %d
outputSurfaceFile: (a file name)
    output file. If unspecified, output file name will be auto
    generated.
    flag: -o %s
```

```
recomputeNormals: (a boolean)
    recompute normals at each iteration
    flag: --norm
searchRadius: (a float, nipy default value: 1)
    search radius
    flag: -r %f
stepSize: (a float, nipy default value: 0.4)
    step size
    flag: -s %f
tangentSmoother: (a float)
    strength of tangential smoother.
    flag: --tc %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timer: (a boolean)
    show timing
    flag: --timer
tissueThreshold: (a float, nipy default value: 1.05)
    tissue threshold
    flag: -t %f
verbosity: (an integer (int or long))
    verbosity
    flag: -v %d
```

Outputs:

```
outputSurfaceFile: (a file name)
    path/name of surface file
```

Pvc

[Link to code](#)**Wraps command `pvc`**

partial volume classifier (PVC) tool. This program performs voxel-wise tissue classification T1-weighted MRI. Image should be skull-stripped and bias-corrected before tissue classification.

<http://brainsuite.org/processing/surfaceextraction/pvc/>

Examples

```
>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> pvc = brainsuite.Pvc()
>>> pvc.inputs.inputMRIFile = example_data('structural.nii')
>>> pvc.inputs.inputMaskFile = example_data('mask.nii')
>>> results = pvc.run()
```

Inputs:

```
[Mandatory]
inputMRIFile: (a file name)
    MRI file
    flag: -i %s

[Optional]
args: (a unicode string)
```



```

        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
inputMaskFile: (a file name)
        brain mask file
        flag: -m %s
outputLabelFile: (a file name)
        output label file. If unspecified, output file name will be auto
        generated.
        flag: -o %s
outputTissueFractionFile: (a file name)
        output tissue fraction file
        flag: -f %s
spatialPrior: (a float)
        spatial prior strength
        flag: -l %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
threeClassFlag: (a boolean)
        use a three-class (CSF=0,GM=1,WM=2) labeling
        flag: -3
timer: (a boolean)
        time processing
        flag: --timer
verbosity: (an integer (int or long))
        verbosity level (0 = silent)
        flag: -v %d

```

Outputs:

```

outputLabelFile: (a file name)
        path/name of label file
outputTissueFractionFile: (a file name)
        path/name of tissue fraction file

```

SVReg[Link to code](#)Wraps command **svreg.sh**

surface and volume registration (svreg) This program registers a subject's BrainSuite-processed volume and surfaces to an atlas, allowing for automatic labelling of volume and surface ROIs.

For more information, please see: <http://brainsuite.org/processing/svreg/usage/>

Examples

```

>>> from nipy.interfaces import brainsuite
>>> svreg = brainsuite.SVReg()
>>> svreg.inputs.subjectFilePrefix = 'home/user/btestsubject/testsubject'
>>> svreg.inputs.refineOutputs = True

```

```
>>> svreg.inputs.skipToVolumeReg = False
>>> svreg.inputs. keepIntermediates = True
>>> svreg.inputs.verbosity2 = True
>>> svreg.inputs.displayTimestamps = True
>>> svreg.inputs.useSingleThreading = True
>>> results = svreg.run()
```

Inputs:

```
[Mandatory]
subjectFilePrefix: (a unicode string)
    Absolute path and filename prefix of the subjects output from
    BrainSuite Cortical Surface Extraction Sequence
    flag: '%s', position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
atlasFilePrefix: (a unicode string)
    Optional: Absolute Path and filename prefix of atlas files and
    labels to which the subject will be registered. If unspecified,
    SVRegwill use its own included atlas files
    flag: '%s', position: 1
curveMatchingInstructions: (a unicode string)
    Used to take control of the curve matching process between the atlas
    and subject. One can specify the name of the .dfc file <sulname.dfc>
    and the sulcal numbers <#sul> to be used as constraints. example:
    curveMatchingInstructions = "subbasename.right.dfc 1 2 20"
    flag: '-cur %s'
dataSinkDelay: (a list of items which are a value of class 'str')
    Connect datasink out_file to dataSinkDelay to delay execution of
    SVReg until dataSink has finished sinking CSE outputs.For use with
    parallel processing workflows including Brainsuites Cortical Surface
    Extraction sequence (SVReg requires certain files from Brainsuite
    CSE, which must all be in the pathway specified by
    subjectFilePrefix. see http://brainsuite.org/processing/svreg/usage/
    for list of required inputs
    flag: %s
displayModuleName: (a boolean)
    Module name will be displayed in the messages
    flag: '-m'
displayTimestamps: (a boolean)
    Timestamps will be displayed in the messages
    flag: '-t'
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
iterations: (an integer (int or long))
    Assigns a number of iterations in the intensity registration step.if
    unspecified, performs 100 iterations
    flag: '-H %d'
keepIntermediates: (a boolean)
    Keep the intermediate files after the svreg sequence is complete.
```

```

    flag: '-k'
pialSurfaceMaskDilation: (an integer (int or long))
    Cortical volume labels found in file output
    subbasename.svreg.label.nii.gz find its boundaries by using the pial
    surface then dilating by 1 voxel. Use this flag in order to control
    the number of pial surface mask dilation. (ie. -D 0 will assign no
    voxel dilation)
    flag: '-D %d'
refineOutputs: (a boolean)
    Refine outputs at the expense of more processing time.
    flag: '-r'
shortMessages: (a boolean)
    Short messages instead of detailed messages
    flag: '-gui'
skipToIntensityReg: (a boolean)
    If the p-harmonic volumetric registration was already performed at
    an earlier time and the user would not like to redo this step, then
    this flag may be used to skip ahead to the intensity registration
    and label transfer step.
    flag: '-p'
skipToVolumeReg: (a boolean)
    If surface registration was already performed at an earlier time and
    the user would not like to redo this step, then this flag may be
    used to skip ahead to the volumetric registration. Necessary input
    files will need to be present in the input directory called by the
    command.
    flag: '-s'
skipVolumetricProcessing: (a boolean)
    Only surface registration and labeling will be performed. Volumetric
    processing will be skipped.
    flag: '-S'
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
useCerebrumMask: (a boolean)
    The cerebrum mask <subbasename.cerebrum.mask.nii.gz> will be used
    for masking the final labels instead of the default pial surface
    mask. Every voxel will be labeled within the cerebrum mask
    regardless of the boundaries of the pial surface.
    flag: '-C'
useManualMaskFile: (a boolean)
    Can call a manually edited cerebrum mask to limit boundaries. Will
    use file: subbasename.cerebrum.mask.nii.gz Make sure to correctly
    replace your manually edited mask file in your input folder with the
    correct subbasename.
    flag: '-cbm'
useMultiThreading: (a boolean)
    If multiple CPUs are present on the system, the code will try to use
    multithreading to make the execution fast.
    flag: '-P'
useSingleThreading: (a boolean)
    Use single threaded mode.
    flag: '-U'
verbosity0: (a boolean)
    no messages will be reported
    flag: '-v0'
    mutually_exclusive: verbosity0, verbosity1, verbosity2

```

```
verbosity1: (a boolean)
    messages will be reported but not the iteration-wise detailed
    messages
    flag: '-v1'
    mutually_exclusive: verbosity0, verbosity1, verbosity2
verbosity2: (a boolean)
    all the messages, including per-iteration, will be displayed
    flag: '-v2'
    mutually_exclusive: verbosity0, verbosity1, verbosity2
```

Outputs:

None

Scrubmask

[Link to code](#)

Wraps command **scrubmask**

ScrubMask tool scrubmask filters binary masks to trim loosely connected voxels that may result from segmentation errors and produce bumps on tessellated surfaces.

<http://brainsuite.org/processing/surfaceextraction/scrubmask/>

Examples

```
>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> scrubmask = brainsuite.Scrubmask()
>>> scrubmask.inputs.inputMaskFile = example_data('mask.nii')
>>> results = scrubmask.run()
```

Inputs:

```
[Mandatory]
inputMaskFile: (a file name)
    input structure mask file
    flag: -i %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
backgroundFillThreshold: (an integer (int or long), nipy default
    value: 2)
    background fill threshold
    flag: -b %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
foregroundTrimThreshold: (an integer (int or long), nipy default
    value: 0)
    foreground trim threshold
    flag: -f %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
numberIterations: (an integer (int or long))
```

```

    number of iterations
    flag: -n %d
outputMaskFile: (a file name)
    output structure mask file. If unspecified, output file name will be
    auto generated.
    flag: -o %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timer: (a boolean)
    timing function
    flag: --timer
verbosity: (an integer (int or long))
    verbosity (0=silent)
    flag: -v %d

```

Outputs:

```

outputMaskFile: (a file name)
    path/name of mask file

```

Skullfinder[Link to code](#)Wraps command **skullfinder**

Skull and scalp segmentation algorithm.

Examples

```

>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data
>>> skullfinder = brainsuite.Skullfinder()
>>> skullfinder.inputs.inputMRIFile = example_data('structural.nii')
>>> skullfinder.inputs.inputMaskFile = example_data('mask.nii')
>>> results = skullfinder.run()

```

Inputs:

```

[Mandatory]
inputMRIFile: (a file name)
    input file
    flag: -i %s
inputMaskFile: (a file name)
    A brain mask file, 8-bit image (0=non-brain, 255=brain)
    flag: -m %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bgLabelValue: (an integer (int or long))
    background label value (0-255)
    flag: --bglabel %d
brainLabelValue: (an integer (int or long))
    brain label value (0-255)
    flag: --brainlabel %d

```

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
         Print an error message instead of throwing an exception in case the
         interface fails to run
lowerThreshold: (an integer (int or long))
         Lower threshold for segmentation
         flag: -l %d
outputLabelFile: (a file name)
         output multi-colored label volume segmenting brain, scalp, inner
         skull & outer skull If unspecified, output file name will be auto
         generated.
         flag: -o %s
performFinalOpening: (a boolean)
         perform a final opening operation on the scalp mask
         flag: --finalOpening
scalpLabelValue: (an integer (int or long))
         scalp label value (0-255)
         flag: --scalplabel %d
skullLabelValue: (an integer (int or long))
         skull label value (0-255)
         flag: --skulllabel %d
spaceLabelValue: (an integer (int or long))
         space label value (0-255)
         flag: --spacelabel %d
surfaceFilePrefix: (a unicode string)
         if specified, generate surface files for brain, skull, and scalp
         flag: -s %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
         Control terminal output: `stream` - displays to terminal immediately
         (default), `allatonce` - waits till command is finished to display
         output, `file` - writes output to file, `none` - output is ignored
upperThreshold: (an integer (int or long))
         Upper threshold for segmentation
         flag: -u %d
verbosity: (an integer (int or long))
         verbosity
         flag: -v %d

```

Outputs:

```

outputLabelFile: (a file name)
         path/name of label file

```

Tca

[Link to code](#)

Wraps command **tca**

topological correction algorithm (TCA) This program removes topological handles from a binary object.

<http://brainsuite.org/processing/surfaceextraction/tca/>

Examples

```

>>> from nipy.interfaces import brainsuite
>>> from nipy.testing import example_data

```

```
>>> tca = brainsuite.Tca()
>>> tca.inputs.inputMaskFile = example_data('mask.nii')
>>> results = tca.run()
```

Inputs:

```
[Mandatory]
inputMaskFile: (a file name)
                input mask volume
                flag: -i %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyre default value: {})
          Environment variables
foregroundDelta: (an integer (int or long), nipyre default value: 20)
                 foreground delta
                 flag: --delta %d
ignore_exception: (a boolean, nipyre default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
maxCorrectionSize: (an integer (int or long))
                   minimum correction size
                   flag: -n %d
minCorrectionSize: (an integer (int or long), nipyre default value:
                   2500)
                   maximum correction size
                   flag: -m %d
outputMaskFile: (a file name)
                 output mask volume. If unspecified, output file name will be auto
                 generated.
                 flag: -o %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
timer: (a boolean)
        timing function
        flag: --timer
verbosity: (an integer (int or long))
            verbosity (0 = quiet)
            flag: -v %d
```

Outputs:

```
outputMaskFile: (a file name)
                 path/name of mask file
```

ThicknessPVC

[Link to code](#)

Wraps command **thicknessPVC.sh**

ThicknessPVC computes cortical thickness using partial tissue fractions. This thickness measure is then transferred to the atlas surface to facilitate population studies. It also stores the computed thickness into separate

hemisphere files and subject thickness mapped to the atlas hemisphere surfaces. ThicknessPVC is not run through the main SVReg sequence, and should be used after executing the BrainSuite and SVReg sequence. For more information, please see:

http://brainsuite.org/processing/svreg/svreg_modules/

Examples

```
>>> from nipy.interfaces import brainsuite
>>> thicknessPVC = brainsuite.ThicknessPVC()
>>> thicknessPVC.inputs.subjectFilePrefix = 'home/user/btestsubject/testsubject'
>>> results = thicknessPVC.run()
```

Inputs:

```
[Mandatory]
subjectFilePrefix: (a unicode string)
    Absolute path and filename prefix of the subject data
    flag: %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
None
```

getFileName()

[Link to code](#)

l_outputs()

[Link to code](#)

interfaces.camino.calib

SFLUTGen

[Link to code](#)

Wraps command **sflutgen**

Generates PICO lookup tables (LUT) for multi-fibre methods such as PASMRI and Q-Ball.

SFLUTGen creates the lookup tables for the generalized multi-fibre implementation of the PICO tractography algorithm. The outputs of this utility are either surface or line coefficients up to a given order. The calibration can be performed for different distributions, such as the Bingham and Watson distributions.

This utility uses calibration data generated from SFPICOCalibData and peak information created by SFPeaks. The utility outputs two lut's, *_oneFibreSurfaceCoeffs.Bdouble and *_twoFibreSurfaceCoeffs.Bdouble. Each of these files contains big- endian doubles as standard. The format of the output is:

```

dimensions      (1 for Watson, 2 for Bingham)
order           (the order of the polynomial)
coefficient_1
coefficient_2
~~~
coefficient_N

```

In the case of the Watson, there is a single set of coefficients, which are ordered:

```
constant, x, x^2, ..., x^order.
```

In the case of the Bingham, there are two sets of coefficients (one for each surface), ordered so that:

```

for j = 1 to order
  for k = 1 to order
    coeff_i = x^j * y^k
  where j+k < order

```

Example

To create a calibration dataset using the default settings

```
>>> import nipyne.interfaces.camino as cam
>>> lutgen = cam.SFLUTGen()
>>> lutgen.inputs.in_file = 'QSH_peaks.Bdouble'
>>> lutgen.inputs.info_file = 'PICO_calib.info'
>>> lutgen.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    Voxel-order data of the spherical functions peaks.
    flag: -inputfile %s
info_file: (a file name)
    The Info file that corresponds to the calibration datafile used in
    the reconstruction.
    flag: -infofile %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
binincsize: (an integer (int or long))
    Sets the size of the bins. In the case of 2D histograms such as the
    Bingham, the bins are always square. Default is 1.
    flag: -binincsize %d
directmap: (a boolean)
    Use direct mapping between the eigenvalues and the distribution
    parameters instead of the log of the eigenvalues.
    flag: -directmap
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
minvectsperbin: (an integer (int or long))
    Specifies the minimum number of fibre-orientation estimates a bin
    must contain before it is used in the lut line/surface generation.
    Default is 50. If you get the error "no fibre-orientation estimates
    in histogram!", the calibration data set is too small to get enough
    samples in any of the histogram bins. You can decrease the minimum
    number per bin to get things running in quick tests, but the sta-
    tistics will not be reliable and for serious applications, you need
    to increase the size of the calibration data set until the error
    goes.
    flag: -minvectsperbin %d
order: (an integer (int or long))
    The order of the polynomial fitting the surface. Order 1 is linear.
    Order 2 (default) is quadratic.
    flag: -order %d
out_file: (a file name)
    flag: > %s, position: -1
outputstem: (a unicode string, nipyne default value: LUT)
    Define the name of the generated luts. The form of the filenames
    will be [outputstem]_oneFibreSurfaceCoeffs.Bdouble and
    [outputstem]_twoFibreSurfaceCoeffs.Bdouble
    flag: -outputstem %s
```

```
pdf: ('bingham' or 'watson', nipy default value: bingham)
    Sets the distribution to use for the calibration. The default is the
    Bingham distribution, which allows elliptical probability density
    contours. Currently supported options are: bingham - The Bingham
    distribution, which allows elliptical probability density contours.
    watson - The Watson distribution. This distribution is rotationally
    symmetric.
    flag: -pdf %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
lut_one_fibre: (an existing file name)
    PICO lut for one-fibre model
lut_two_fibres: (an existing file name)
    PICO lut for two-fibre model
```

SFPICOCalibData[Link to code](#)Wraps command **sfpicocalibdata**

Generates Spherical Function PICO Calibration Data.

SFPICOCalibData creates synthetic data for use with SFLUTGen. The synthetic data is generated using a mixture of gaussians, in the same way datasynth generates data. Each voxel of data models a slightly different fibre configuration (varying FA and fibre- crossings) and undergoes a random rotation to help account for any directional bias in the chosen acquisition scheme. A second file, which stores information about the datafile, is generated along with the datafile.

Example 1

To create a calibration dataset using the default settings

```
>>> import nipy.interfaces.camino as cam
>>> calib = cam.SFPICOCalibData()
>>> calib.inputs.scheme_file = 'A.scheme'
>>> calib.inputs.snr = 20
>>> calib.inputs.info_file = 'PICO_calib.info'
>>> calib.run()
```

The default settings create a large dataset (249,231 voxels), of which 3401 voxels contain a single fibre population per voxel and the rest of the voxels contain two fibre-populations. The amount of data produced can be varied by specifying the ranges and steps of the parameters for both the one and two fibre datasets used.

Example 2

To create a custom calibration dataset

```
>>> import nipy.interfaces.camino as cam
>>> calib = cam.SFPICOCalibData()
>>> calib.inputs.scheme_file = 'A.scheme'
>>> calib.inputs.snr = 20
>>> calib.inputs.info_file = 'PICO_calib.info'
>>> calib.inputs.twodtfarange = [0.3, 0.9]
>>> calib.inputs.twodtfastep = 0.02
>>> calib.inputs.twodtanglerange = [0, 0.785]
```

```
>>> calib.inputs.twodtanglestep = 0.03925
>>> calib.inputs.twodtmixmax = 0.8
>>> calib.inputs.twodtmixstep = 0.1
>>> calib.run()
```

This would provide 76,313 voxels of synthetic data, where 3401 voxels simulate the one fibre cases and 72,912 voxels simulate the various two fibre cases. However, care should be taken to ensure that enough data is generated for calculating the LUT. # doctest: +SKIP

Inputs:

```
[Mandatory]
info_file: (a file name)
    The name to be given to the information output filename.
    flag: -infooutputfile %s
scheme_file: (an existing file name)
    Specifies the scheme file for the diffusion MRI data
    flag: -schemefile %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
onedtfarange: (a list of from 2 to 2 items which are a float)
    Minimum and maximum FA for the single tensor synthetic data.
    flag: -onedtfarange %s
onedtfastep: (a float)
    FA step size controlling how many steps there are between the
    minimum and maximum FA settings.
    flag: -onedtfastep %f
out_file: (a file name)
    flag: > %s, position: -1
seed: (a float)
    Specifies the random seed to use for noise generation in simulation
    trials.
    flag: -seed %f
snr: (a float)
    Specifies the signal-to-noise ratio of the non-diffusion-weighted
    measurements to use in simulations.
    flag: -snr %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
trace: (a float)
    Trace of the diffusion tensor(s) used in the test function.
    flag: -trace %f
twodtanglerange: (a list of from 2 to 2 items which are a float)
    Minimum and maximum crossing angles between the two fibres.
    flag: -twodtanglerange %s
twodtanglestep: (a float)
    Angle step size controlling how many steps there are between the
```

```

        minimum and maximum crossing angles for the two tensor cases.
        flag: -twodtanglestep %f
twodtfarange: (a list of from 2 to 2 items which are a float)
        Minimum and maximum FA for the two tensor synthetic data. FA is
        varied for both tensors to give all the different permutations.
        flag: -twodtfarange %s
twodtfastep: (a float)
        FA step size controlling how many steps there are between the
        minimum and maximum FA settings for the two tensor cases.
        flag: -twodtfastep %f
twodtmixmax: (a float)
        Mixing parameter controlling the proportion of one fibre population
        to the other. The minimum mixing parameter is (1 - twodtmixmax).
        flag: -twodtmixmax %f
twodtmixstep: (a float)
        Mixing parameter step size for the two tensor cases. Specify how
        many mixing parameter increments to use.
        flag: -twodtmixstep %f

```

Outputs:

```

PICOCalib: (an existing file name)
    Calibration dataset
calib_info: (an existing file name)
    Calibration dataset

```

interfaces.camino.connectivity

Conmat

[Link to code](#)Wraps command **conmat**

Creates a connectivity matrix using a 3D label image (the target image) and a set of streamlines. The connectivity matrix records how many stream- lines connect each pair of targets, and optionally the mean tractwise statistic (eg tract-averaged FA, or length).

The output is a comma separated variable file or files. The first row of the output matrix is label names. Label names may be defined by the user, otherwise they are assigned based on label intensity.

Starting from the seed point, we move along the streamline until we find a point in a labeled region. This is done in both directions from the seed point. Streamlines are counted if they connect two target regions, one on either side of the seed point. Only the labeled region closest to the seed is counted, for example if the input contains two streamlines:

```

1: A-----B-----SEED---C
2: A-----SEED-----

```

then the output would be

```

A,B,C
0,0,0
0,0,1
0,1,0

```

There are zero connections to A because in streamline 1, the connection to B is closer to the seed than the connection to A, and in streamline 2 there is no region reached in the other direction.

The connected target regions can have the same label, as long as the seed point is outside of the labeled region and both ends connect to the same label (which may be in different locations). Therefore this is allowed:

```
A-----SEED-----A
```

Such fibers will add to the diagonal elements of the matrix. To remove these entries, run `procstreamlines` with `-endpointfile` before running `conmat`.

If the seed point is inside a labeled region, it counts as one end of the connection. So

```
----[SEED inside A]-----B
```

counts as a connection between A and B, while

```
C----[SEED inside A]-----B
```

counts as a connection between A and C, because C is closer to the seed point.

In all cases, distance to the seed point is defined along the streamline path.

Example 1

To create a standard connectivity matrix based on streamline counts.

```
>>> import nipy.interfaces.camino as cam
>>> conmat = cam.Conmat()
>>> conmat.inputs.in_file = 'tracts.Bdouble'
>>> conmat.inputs.target_file = 'atlas.nii.gz'
>>> conmat.run()
```

Example 1

To create a standard connectivity matrix and mean tractwise FA statistics.

```
>>> import nipy.interfaces.camino as cam
>>> conmat = cam.Conmat()
>>> conmat.inputs.in_file = 'tracts.Bdouble'
>>> conmat.inputs.target_file = 'atlas.nii.gz'
>>> conmat.inputs.scalar_file = 'fa.nii.gz'
>>> conmat.tract_stat = 'mean'
>>> conmat.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Streamlines as generated by the Track interface
        flag: -inputfile %s
target_file: (an existing file name)
        An image containing targets, as used in ProcStreamlines interface.
        flag: -targetfile %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
output_root: (a file name)
```

```

        filename root prepended onto the names of the output files. The
        extension will be determined from the input.
        flag: -outputroot %s
scalar_file: (an existing file name)
        Optional scalar file for computing tract-based statistics. Must be
        in the same space as the target file.
        flag: -scalarfile %s
        requires: tract_stat
targetname_file: (an existing file name)
        Optional names of targets. This file should contain one entry per
        line, with the target intensity followed by the name, separated by
        white space. For example: 1 some_brain_region 2 some_other_region
        These names will be used in the output. The names themselves should
        not contain spaces or commas. The labels may be in any order but the
        output matrices will be ordered by label intensity.
        flag: -targetnamefile %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
tract_prop: ('length' or 'endpointsep')
        Tract property average to compute in the connectivity matrix. See
        TractStats for details.
        flag: -tractstat %s
        mutually_exclusive: tract_stat
tract_stat: ('mean' or 'min' or 'max' or 'sum' or 'median' or 'var')
        Tract statistic to use. See TractStats for other options.
        flag: -tractstat %s
        mutually_exclusive: tract_prop
        requires: scalar_file

```

Outputs:

```

conmat_sc: (an existing file name)
        Connectivity matrix in CSV file.
conmat_ts: (a file name)
        Tract statistics in CSV file.

```

interfaces.camino.convert

AnalyzeHeader

[Link to code](#)Wraps command **analyzeheader**

Create or read an Analyze 7.5 header file.

Analyze image header, provides support for the most common header fields. Some fields, such as patient_id, are not currently supported. The program allows three nonstandard options: the field image_dimension.funused1 is the image scale. The intensity of each pixel in the associated .img file is (image value from file) * scale. Also, the origin of the Talairach coordinates (midline of the anterior commissure) are encoded in the field data_history.originator. These changes are included for compatibility with SPM.

All headers written with this program are big endian by default.

Example

```
>>> import nipyype.interfaces.camino as cmon
>>> hdr = cmon.AnalyzeHeader()
>>> hdr.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> hdr.inputs.scheme_file = 'A.scheme'
>>> hdr.inputs.data_dims = [256,256,256]
>>> hdr.inputs.voxel_dims = [1,1,1]
>>> hdr.run()
```

Inputs:

```
[Mandatory]
datatype: ('byte' or 'char' or '[u]short' or '[u]int' or 'float' or
          'complex' or 'double')
    The char datatype is 8 bit (not the 16 bit char of Java), as
    specified by the Analyze 7.5 standard. The byte, ushort and uint
    types are not part of the Analyze specification but are supported by
    SPM.
    flag: -datatype %s
in_file: (an existing file name)
    Tensor-fitted data filename
    flag: < %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
centre: (a list of from 3 to 3 items which are an integer (int or
        long))
    Voxel specifying origin of Talairach coordinate system for SPM,
    default [0 0 0].
    flag: -centre %s
data_dims: (a list of from 3 to 3 items which are an integer (int or
            long))
    data dimensions in voxels
    flag: -datadims %s
description: (a string)
    Short description - No spaces, max length 79 bytes. Will be null
    terminated automatically.
    flag: -description %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyype default value: {})
    Environment variables
greylevels: (a list of from 2 to 2 items which are an integer (int or
            long))
    Minimum and maximum greylevels. Stored as shorts in the header.
    flag: -gl %s
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initfromheader: (an existing file name)
    Reads header information from file and initializes a new header with
    the values read from the file. You may replace any combination of
    fields in the new header by specifying subsequent options.
    flag: -initfromheader %s, position: 3
intelbyteorder: (a boolean)
    Write header in intel byte order (little-endian).
```



```

    flag: -intelbyteorder
networkbyteorder: (a boolean)
    Write header in network byte order (big-endian). This is the default
    for new headers.
    flag: -networkbyteorder
nimages: (an integer (int or long))
    Number of images in the img file. Default 1.
    flag: -nimages %d
offset: (an integer (int or long))
    According to the Analyze 7.5 standard, this is the byte offset in
    the .img file at which voxels start. This value can be negative to
    specify that the absolute value is applied for every image in the
    file.
    flag: -offset %d
out_file: (a file name)
    flag: > %s, position: -1
picoseed: (a list of from 3 to 3 items which are an integer (int or
    long))
    Voxel specifying the seed (for PICO maps), default [0 0 0].
    flag: -picoseed %s
printbigendian: (an existing file name)
    Prints 1 if the header is big-endian, 0 otherwise.
    flag: -printbigendian %s, position: 3
printimagedims: (an existing file name)
    Prints image data and voxel dimensions as Camino arguments and
    exits.
    flag: -printimagedims %s, position: 3
printintelbyteorder: (an existing file name)
    Prints 1 if the header is little-endian, 0 otherwise.
    flag: -printintelbyteorder %s, position: 3
printprogargs: (an existing file name)
    Prints data dimension (and type, if relevant) arguments for a
    specific Camino program, where prog is one of shredder,
    scanner2voxel, vctreshselect, pdview, track.
    flag: -printprogargs %s, position: 3
readheader: (an existing file name)
    Reads header information from file and prints to stdout. If this
    option is not specified, then the program writes a header based on
    the other arguments.
    flag: -readheader %s, position: 3
scaleinter: (a float)
    Constant to add to the image intensities. Used by SPM and MRICro.
    flag: -scaleinter %d
scaleslope: (a float)
    Intensities in the image are scaled by this factor by SPM and
    MRICro. Default is 1.0.
    flag: -scaleslope %d
scheme_file: (an existing file name)
    Camino scheme file (b values / vectors, see camino.fsl2scheme)
    flag: %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    flag: -voxeldims %s

```

Outputs:

```
header: (an existing file name)
        Analyze header
```

DT2NiftI

[Link to code](#)

Wraps command **dt2nii**

Converts camino tensor data to NiftI format

Reads Camino diffusion tensors, and converts them to NIFTI format as three .nii files.

Inputs:

```
[Mandatory]
header_file: (an existing file name)
              A Nifti .nii or .hdr file containing the header information
              flag: -header %s, position: 3
in_file: (an existing file name)
          tract file
          flag: -inputfile %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipype default value: {})
          Environment variables
ignore_exception: (a boolean, nipype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
output_root: (a file name)
              filename root prepended onto the names of three output files.
              flag: -outputroot %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
dt: (an existing file name)
    diffusion tensors in NiftI format
exitcode: (an existing file name)
          exit codes from Camino reconstruction in NiftI format
lns0: (an existing file name)
      estimated lns0 from Camino reconstruction in NiftI format
```

Image2Voxel

[Link to code](#)

Wraps command **image2voxel**

Converts Analyze / NIFTI / MHA files to voxel order.

Converts scanner-order data in a supported image format to voxel-order data. Either takes a 4D file (all measurements in single image) or a list of 3D images.

Examples

```
>>> import nipy.interfaces.camino as cmon
>>> img2vox = cmon.Image2Voxel()
>>> img2vox.inputs.in_file = '4d_dwi.nii'
>>> img2vox.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        4d image file
        flag: -4dimage %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          flag: > %s, position: -1
out_type: ('float' or 'char' or 'short' or 'int' or 'long' or
           'double', nipy default value: float)
           "i.e. Bfloat". Can be "char", "short", "int", "long", "float" or
           "double"
           flag: -outputdatatype %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
voxel_order: (an existing file name)
             path/name of 4D volume in voxel order
```

Nifti2Camino

[Link to code](#)

Wraps command **nifti2camino**

Converts NIFTI-1 diffusion tensors to Camino format. The program reads the NIFTI header but does not apply any spatial transformations to the data. The NIFTI intensity scaling parameters are applied.

The output is the tensors in Camino voxel ordering: [exit, ln(S0), dxx, dxy, dxz, dyx, dyz, dzz].

The exit code is set to 0 unless a background mask is supplied, in which case the code is 0 in brain voxels and -1 in background voxels.

The value of ln(S0) in the output is taken from a file if one is supplied, otherwise it is set to 0.

NOTE FOR FSL USERS - FSL's dtfit can output NIFTI tensors, but they are not stored in the usual way (which is using NIFTI_INTENT_SYMMATRIX). FSL's tensors follow the ITK / VTK "upper-triangular" convention, so you will need to use the -uppertriangular option to convert these correctly.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    A NIFTI-1 dataset containing diffusion tensors. The tensors are
    assumed to be in lower-triangular order as specified by the NIFTI
    standard for the storage of symmetric matrices. This file should be
    either a .nii or a .hdr file.
    flag: -inputfile %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bgmask: (an existing file name)
    Binary valued brain / background segmentation, may be a raw binary
    file (specify type with -maskdatatype) or a supported image file.
    flag: -bgmask %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
lns0_file: (an existing file name)
    File containing the log of the unweighted signal for each voxel, may
    be a raw binary file (specify type with -inputdatatype) or a
    supported image file.
    flag: -lns0 %s
out_file: (a file name)
    flag: > %s, position: -1
s0_file: (an existing file name)
    File containing the unweighted signal for each voxel, may be a raw
    binary file (specify type with -inputdatatype) or a supported image
    file.
    flag: -s0 %s
scaleinter: (a float)
    A value v in the diffusion tensor is scaled to  $v * s + i$ . This is
    applied after any scaling specified by the input image. Default is
    0.0.
    flag: -scaleinter %s
scaleslope: (a float)
    A value v in the diffusion tensor is scaled to  $v * s + i$ . This is
    applied after any scaling specified by the input image. Default is
    1.0.
    flag: -scaleslope %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
uppertriangular: (a boolean)
    Specifies input in upper-triangular (VTK style) order.
    flag: -uppertriangular %s

```

Outputs:

```

out_file: (a file name)
    diffusion tensors data in Camino format

```

ProcStreamlines

[Link to code](#)

Wraps command **procstreamlines**

Process streamline data

This program does post-processing of streamline output from track. It can either output streamlines or connection probabilities.

<http://web4.cs.ucl.ac.uk/research/medic/camino/pmwiki/pmwiki.php?n=Man.procstreamlines>

Examples

```
>>> import nipy.interfaces.camino as cmon
>>> proc = cmon.ProcStreamlines()
>>> proc.inputs.in_file = 'tract_data.Bfloat'
>>> proc.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        data file
        flag: -inputfile %s, position: 1

[Optional]
allowmultitargets: (a boolean)
                  Allows streamlines to connect to multiple target volumes.
                  flag: -allowmultitargets
args: (a unicode string)
      Additional parameters to the command
      flag: %s
datadims: (a list of from 3 to 3 items which are an integer (int or
          long))
          data dimensions in voxels
          flag: -datadims %s
directional: (a list of from 3 to 3 items which are an integer (int
            or long))
            Splits the streamlines at the seed point and computes separate
            connection probabilities for each segment. Streamline segments are
            grouped according to their dot product with the vector (X, Y, Z).
            The ideal vector will be tangential to the streamline trajectory at
            the seed, such that the streamline projects from the seed along (X,
            Y, Z) and -(X, Y, Z). However, it is only necessary for the
            streamline trajectory to not be orthogonal to (X, Y, Z).
            flag: -directional %s
discardloops: (a boolean)
              This option allows streamlines to enter a waypoint exactly once.
              After the streamline leaves the waypoint, the entire streamline is
              discarded upon a second entry to the waypoint.
              flag: -discardloops
endpointfile: (a file name)
              Image containing endpoint ROIs. This should be an Analyze 7.5 header
              / image file.hdr and file.img.
              flag: -endpointfile %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
exclusionfile: (a file name)
```

```
Image containing exclusion ROIs. This should be an Analyze 7.5
header / image file.hdr and file.img.
flag: -exclusionfile %s
gzip: (a boolean)
    save the output image in gzip format
    flag: -gzip
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputmodel: ('raw' or 'voxels', nipy default value: raw)
    input model type (raw or voxels)
    flag: -inputmodel %s
iterations: (a float)
    Number of streamlines generated for each seed. Not required when
    outputting streamlines, but needed to create PICO images. The
    default is 1 if the output is streamlines, and 5000 if the output is
    connection probability images.
    flag: -iterations %d
maxtractlength: (an integer (int or long))
    maximum length of tracts
    flag: -maxtractlength %d
maxtractpoints: (an integer (int or long))
    maximum number of tract points
    flag: -maxtractpoints %d
mintractlength: (an integer (int or long))
    minimum length of tracts
    flag: -mintractlength %d
mintractpoints: (an integer (int or long))
    minimum number of tract points
    flag: -mintractpoints %d
noresample: (a boolean)
    Disables resampling of input streamlines. Resampling is
    automatically disabled if the input model is voxels.
    flag: -noresample
out_file: (a file name)
    flag: > %s, position: -1
outputacm: (a boolean)
    output all tracts in a single connection probability map (Analyze
    image)
    flag: -outputacm
    requires: outputroot, seedfile
outputcbs: (a boolean)
    outputs connectivity-based segmentation maps; requires target
    outputfile
    flag: -outputcbs
    requires: outputroot, targetfile, seedfile
outputcp: (a boolean)
    output the connection probability map (Analyze image, float)
    flag: -outputcp
    requires: outputroot, seedfile
outputroot: (a file name)
    Prepended onto all output file names.
    flag: -outputroot %s
outputsc: (a boolean)
    output the connection probability map (raw streamlines, int)
    flag: -outputsc
    requires: outputroot, seedfile
outputtracts: (a boolean)
```

```

    Output streamlines in raw binary format.
    flag: -outputtracts
regionindex: (an integer (int or long))
    index of specific region to process
    flag: -regionindex %d
resamplestepsize: (a float)
    Each point on a streamline is tested for entry into target,
    exclusion or waypoint volumes. If the length between points on a
    tract is not much smaller than the voxel length, then streamlines
    may pass through part of a voxel without being counted. To avoid
    this, the program resamples streamlines such that the step size is
    one tenth of the smallest voxel dimension in the image. This
    increases the size of raw or oogl streamline output and incurs some
    performance penalty. The resample resolution can be controlled with
    this option or disabled altogether by passing a negative step size
    or by passing the -noresample option.
    flag: -resamplestepsize %d
seedfile: (a file name)
    Image Containing Seed Points
    flag: -seedfile %s
seedpointmm: (a list of from 3 to 3 items which are an integer (int
    or long))
    The coordinates of a single seed point for tractography in mm
    flag: -seedpointmm %s
seedpointvox: (a list of from 3 to 3 items which are an integer (int
    or long))
    The coordinates of a single seed point for tractography in voxels
    flag: -seedpointvox %s
targetfile: (a file name)
    Image containing target volumes.
    flag: -targetfile %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
truncateinexclusion: (a boolean)
    Retain segments of a streamline before entry to an exclusion ROI.
    flag: -truncateinexclusion
truncateloops: (a boolean)
    This option allows streamlines to enter a waypoint exactly once.
    After the streamline leaves the waypoint, it is truncated upon a
    second entry to the waypoint.
    flag: -truncateloops
voxeldims: (a list of from 3 to 3 items which are an integer (int or
    long))
    voxel dimensions in mm
    flag: -voxeldims %s
waypointfile: (a file name)
    Image containing waypoints. Waypoints are defined as regions of the
    image with the same intensity, where 0 is background and any value >
    0 is a waypoint.
    flag: -waypointfile %s

```

Outputs:

```

outputroot_files: (a list of items which are an existing file name)
proc: (an existing file name)
    Processed Streamlines

```

Shredder

[Link to code](#)

Wraps command **shredder**

Extracts periodic chunks from a data stream.

Shredder makes an initial offset of offset bytes. It then reads and outputs chunksize bytes, skips space bytes, and repeats until there is no more input.

If the chunksize is negative, chunks of size chunksize are read and the byte ordering of each chunk is reversed. The whole chunk will be reversed, so the chunk must be the same size as the data type, otherwise the order of the values in the chunk, as well as their endianness, will be reversed.

Examples

```
>>> import nipyre.interfaces.camino as cam
>>> shred = cam.Shredder()
>>> shred.inputs.in_file = 'SubjectA.Bfloat'
>>> shred.inputs.offset = 0
>>> shred.inputs.chunksize = 1
>>> shred.inputs.space = 2
>>> shred.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        raw binary data file
        flag: < %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
chunksize: (an integer (int or long))
           reads and outputs a chunk of chunksize bytes
           flag: %d, position: 2
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipyre default value: {})
         Environment variables
ignore_exception: (a boolean, nipyre default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
offset: (an integer (int or long))
        initial offset of offset bytes
        flag: %d, position: 1
out_file: (a file name)
          flag: > %s, position: -1
space: (an integer (int or long))
       skips space bytes
       flag: %d, position: 3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:


```
shredded: (an existing file name)
          Shredded binary data file
```

TractShredder

[Link to code](#)

Wraps command **tractshredder**

Extracts bunches of streamlines.

tractshredder works in a similar way to shredder, but processes streamlines instead of scalar data. The input is raw streamlines, in the format produced by track or proctstreamlines.

The program first makes an initial offset of offset tracts. It then reads and outputs a group of bunchsize tracts, skips space tracts, and repeats until there is no more input.

Examples

```
>>> import nipy.interfaces.camino as cmon
>>> shred = cmon.TractShredder()
>>> shred.inputs.in_file = 'tract_data.Bfloat'
>>> shred.inputs.offset = 0
>>> shred.inputs.bunchsize = 1
>>> shred.inputs.space = 2
>>> shred.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         tract file
         flag: < %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
bunchsize: (an integer (int or long))
            reads and outputs a group of bunchsize tracts
            flag: %d, position: 2
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
offset: (an integer (int or long))
        initial offset of offset tracts
        flag: %d, position: 1
out_file: (a file name)
          flag: > %s, position: -1
space: (an integer (int or long))
        skips space tracts
        flag: %d, position: 3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
shredded: (an existing file name)
    Shredded tract file
```

VtkStreamlines

[Link to code](#)

Wraps command **vtkstreamlines**

Use **vtkstreamlines** to convert raw or voxel format streamlines to VTK polydata

Examples

```
>>> import nipy.interfaces.camino as cmon
>>> vtk = cmon.VtkStreamlines()
>>> vtk.inputs.in_file = 'tract_data.Bfloat'
>>> vtk.inputs.voxeldims = [1,1,1]
>>> vtk.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    data file
    flag: < %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
colourorient: (a boolean)
    Each point on the streamline is coloured by the local orientation.
    flag: -colourorient
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputmodel: ('raw' or 'voxels', nipy default value: raw)
    input model type (raw or voxels)
    flag: -inputmodel %s
interpolate: (a boolean)
    the scalar value at each point on the streamline is calculated by
    trilinear interpolation
    flag: -interpolate
interpolatescalars: (a boolean)
    the scalar value at each point on the streamline is calculated by
    trilinear interpolation
    flag: -interpolatescalars
out_file: (a file name)
    flag: > %s, position: -1
scalar_file: (a file name)
    image that is in the same physical space as the tracts
    flag: -scalarfile %s, position: 3
seed_file: (a file name)
```

```

        image containing seed points
        flag: -seedfile %s, position: 1
target_file: (a file name)
        image containing integer-valued target regions
        flag: -targetfile %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
voxeldims: (a list of from 3 to 3 items which are an integer (int or
        long))
        voxel dimensions in mm
        flag: -voxeldims %s, position: 4

```

Outputs:

```

vtk: (an existing file name)
        Streamlines in VTK format

```

interfaces.camino.dti

ComputeEigensystem

[Link to code](#)

Wraps command **dteig**

Computes the eigensystem from tensor fitted data.

Reads diffusion tensor (single, two-tensor, three-tensor or multitensor) data from the standard input, computes the eigenvalues and eigenvectors of each tensor and outputs the results to the standard output. For multiple-tensor data the program outputs the eigensystem of each tensor. For each tensor the program outputs: {l_1, e_11, e_12, e_13, l_2, e_21, e_22, e_33, l_3, e_31, e_32, e_33}, where $l_1 \geq l_2 \geq l_3$ and $e_i = (e_{i1}, e_{i2}, e_{i3})$ is the eigenvector with eigenvalue l_i . For three-tensor data, for example, the output contains thirty-six values per voxel.

Example

```

>>> import nipy.interfaces.camino as cmon
>>> dteig = cmon.ComputeEigensystem()
>>> dteig.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> dteig.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        Tensor-fitted data filename
        flag: < %s, position: 1

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)

```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
inputdatatype: ('double' or 'float' or 'long' or 'int' or 'short' or
               'char', nipy default value: double)
        Specifies the data type of the input data. The data type can be any
        of the following strings: "char", "short", "int", "long", "float" or
        "double".Default is double data type
        flag: -inputdatatype %s
inputmodel: ('dt' or 'multitensor')
        Specifies the model that the input data contains parameters for.
        Possible model types are: "dt" (diffusion-tensor data) and
        "multitensor"
        flag: -inputmodel %s
maxcomponents: (an integer (int or long))
        The maximum number of tensor components in a voxel of the input
        data.
        flag: -maxcomponents %d
out_file: (a file name)
        flag: > %s, position: -1
outputdatatype: ('double' or 'float' or 'long' or 'int' or 'short' or
                'char', nipy default value: double)
        Specifies the data type of the output data. The data type can be any
        of the following strings: "char", "short", "int", "long", "float" or
        "double".Default is double data type
        flag: -outputdatatype %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

eigen: (an existing file name)
        Trace of the diffusion tensor

```

ComputeFractionalAnisotropy

[Link to code](#)Wraps command **fa**

Computes the fractional anisotropy of tensors.

Reads diffusion tensor (single, two-tensor or three-tensor) data from the standard input, computes the fractional anisotropy (FA) of each tensor and outputs the results to the standard output. For multiple-tensor data the program outputs the FA of each tensor, so for three-tensor data, for example, the output contains three fractional anisotropy values per voxel.

Example

```

>>> import nipy.interfaces.camino as cmon
>>> fa = cmon.ComputeFractionalAnisotropy()
>>> fa.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> fa.inputs.scheme_file = 'A.scheme'
>>> fa.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)

```

```

    Tensor-fitted data filename
    flag: < %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or
    'double')
    Specifies the data type of the input file. The data type can be any
    of thefollowing strings: "char", "short", "int", "long", "float" or
    "double".
    flag: -inputdatatype %s
inputmodel: ('dt' or 'twotensor' or 'threetensor' or 'multitensor')
    Specifies the model that the input tensor data contains parameters
    for.Possible model types are: "dt" (diffusion-tensor data),
    "twotensor" (two-tensor data), "threetensor" (three-tensor data). By
    default, the program assumes that the input data contains a single
    diffusion tensor in each voxel.
    flag: -inputmodel %s
out_file: (a file name)
    flag: > %s, position: -1
outputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or
    'double')
    Specifies the data type of the output data. The data type can be any
    of thefollowing strings: "char", "short", "int", "long", "float" or
    "double".
    flag: -outputdatatype %s
scheme_file: (an existing file name)
    Camino scheme file (b values / vectors, see camino.fsl2scheme)
    flag: %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

fa: (an existing file name)
    Fractional Anisotropy Map

```

ComputeMeanDiffusivity[Link to code](#)Wraps command **md**

Computes the mean diffusivity (trace/3) from diffusion tensors.

Example

```
>>> import nipy.interfaces.camino as cmon
>>> md = cmon.ComputeMeanDiffusivity()
>>> md.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> md.inputs.scheme_file = 'A.scheme'
>>> md.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Tensor-fitted data filename
        flag: < %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
inputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or
                'double')
                Specifies the data type of the input file. The data type can be any
                of the following strings: "char", "short", "int", "long", "float" or
                "double".
                flag: -inputdatatype %s
inputmodel: ('dt' or 'twotensor' or 'threetensor')
             Specifies the model that the input tensor data contains parameters
             for. Possible model types are: "dt" (diffusion-tensor data),
             "twotensor" (two-tensor data), "threetensor" (three-tensor data). By
             default, the program assumes that the input data contains a single
             diffusion tensor in each voxel.
             flag: -inputmodel %s
out_file: (a file name)
          flag: > %s, position: -1
outputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or
                 'double')
                 Specifies the data type of the output data. The data type can be any
                 of the following strings: "char", "short", "int", "long", "float" or
                 "double".
                 flag: -outputdatatype %s
scheme_file: (an existing file name)
             Camino scheme file (b values / vectors, see camino.fsl2scheme)
             flag: %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
md: (an existing file name)
    Mean Diffusivity Map
```

ComputeTensorTrace

[Link to code](#)

Wraps command **trd**

Computes the trace of tensors.

Reads diffusion tensor (single, two-tensor or three-tensor) data from the standard input, computes the trace of each tensor, i.e., three times the mean diffusivity, and outputs the results to the standard output. For multiple-tensor data the program outputs the trace of each tensor, so for three-tensor data, for example, the output contains three values per voxel.

Divide the output by three to get the mean diffusivity.

Example

```
>>> import nipy.interfaces.camino as cmon
>>> trace = cmon.ComputeTensorTrace()
>>> trace.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> trace.inputs.scheme_file = 'A.scheme'
>>> trace.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Tensor-fitted data filename
        flag: < %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
inputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or
               'double')
               Specifies the data type of the input file. The data type can be any
               of the following strings: "char", "short", "int", "long", "float" or
               "double".
               flag: -inputdatatype %s
inputmodel: ('dt' or 'twotensor' or 'threetensor' or 'multitensor')
             Specifies the model that the input tensor data contains parameters
             for. Possible model types are: "dt" (diffusion-tensor data),
             "twotensor" (two-tensor data), "threetensor" (three-tensor data). By
             default, the program assumes that the input data contains a single
             diffusion tensor in each voxel.
             flag: -inputmodel %s
out_file: (a file name)
          flag: > %s, position: -1
outputdatatype: ('char' or 'short' or 'int' or 'long' or 'float' or
                'double')
                Specifies the data type of the output data. The data type can be any
                of the following strings: "char", "short", "int", "long", "float" or
                "double".
                flag: -outputdatatype %s
```

```
scheme_file: (an existing file name)
    Camino scheme file (b values / vectors, see camino.fsl2scheme)
    flag: %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
trace: (an existing file name)
    Trace of the diffusion tensor
```

DTIFit

[Link to code](#)Wraps command **dtfit**

Reads diffusion MRI data, acquired using the acquisition scheme detailed in the scheme file, from the data file. Use non-linear fitting instead of the default linear regression to the log measurements. The data file stores the diffusion MRI data in voxel order with the measurements stored in big-endian format and ordered as in the scheme file. The default input data type is four-byte float. The default output data type is eight-byte double. See `modelfit` and `camino` for the format of the data file and scheme file. The program fits the diffusion tensor to each voxel and outputs the results, in voxel order and as big-endian eight-byte doubles, to the standard output. The program outputs eight values in each voxel: [exit code, $\ln(S(0))$, D_{xx} , D_{xy} , D_{xz} , D_{yy} , D_{yz} , D_{zz}]. An exit code of zero indicates no problems. For a list of other exit codes, see `modelfit(1)`. The entry $S(0)$ is an estimate of the signal at $q=0$.

Example

```
>>> import nipy.interfaces.camino as cmon
>>> fit = cmon.DTIFit()
>>> fit.inputs.scheme_file = 'A.scheme'
>>> fit.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> fit.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    voxel-order data filename
    flag: %s, position: 1
scheme_file: (an existing file name)
    Camino scheme file (b values / vectors, see camino.fsl2scheme)
    flag: %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bgmask: (an existing file name)
    Provides the name of a file containing a background mask computed
    using, for example, FSL bet2 program. The mask file contains zero in
    background voxels and non-zero in foreground.
    flag: -bgmask %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
```



```

    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
non_linear: (a boolean)
    Use non-linear fitting instead of the default linear regression to
    the log measurements.
    flag: -nonlinear, position: 3
out_file: (a file name)
    flag: > %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

tensor_fitted: (an existing file name)
    path/name of 4D volume in voxel order

```

DTLUTGen[Link to code](#)Wraps command **dtlutgen**

Calibrates the PDFs for PICO probabilistic tractography.

This program needs to be run once for every acquisition scheme. It outputs a lookup table that is used by the dtpicoparams program to find PICO PDF parameters for an image. The default single tensor LUT contains parameters of the Bingham distribution and is generated by supplying a scheme file and an estimated signal to noise in white matter regions of the (q=0) image. The default inversion is linear (inversion index 1).

Advanced users can control several options, including the extent and resolution of the LUT, the inversion index, and the type of PDF. See dtlutgen(1) for details.

Example

```

>>> import nipy.interfaces.camino as cmon
>>> dtl = cmon.DTLUTGen()
>>> dtl.inputs.snr = 16
>>> dtl.inputs.scheme_file = 'A.scheme'
>>> dtl.run()

```

Inputs:

```

[Mandatory]
scheme_file: (a file name)
    The scheme file of the images to be processed using this LUT.
    flag: -schemefile %s, position: 2

[Optional]
acg: (a boolean)
    Compute a LUT for the ACG PDF.
    flag: -acg
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bingham: (a boolean)
    Compute a LUT for the Bingham PDF. This is the default.
    flag: -bingham

```

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
frange: (a list of from 2 to 2 items which are a float)
         Index to two-tensor LUTs. This is the fractional anisotropy of the
         two tensors. The default is 0.3 to 0.94
         flag: -frange %s, position: 1
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
inversion: (an integer (int or long))
            Index of the inversion to use. The default is 1 (linear single
            tensor inversion).
            flag: -inversion %d
lrange: (a list of from 2 to 2 items which are a float)
         Index to one-tensor LUTs. This is the ratio L1/L3 and L2 / L3. The
         LUT is square, with half the values calculated (because L2 / L3
         cannot be less than L1 / L3 by definition). The minimum must be >= 1.
         For comparison, a ratio L1 / L3 = 10 with L2 / L3 = 1 corresponds to
         an FA of 0.891, and L1 / L3 = 15 with L2 / L3 = 1 corresponds to an
         FA of 0.929. The default range is 1 to 10.
         flag: -lrange %s, position: 1
out_file: (a file name)
           flag: > %s, position: -1
samples: (an integer (int or long))
          The number of synthetic measurements to generate at each point in
          the LUT. The default is 2000.
          flag: -samples %d
snr: (a float)
      The signal to noise ratio of the unweighted (q = 0)
      measurements. This should match the SNR (in white matter) of the
      images that the LUTs are used with.
      flag: -snr %f
step: (a float)
       Distance between points in the LUT. For example, if lrange is 1 to 10
       and the step is 0.1, LUT entries will be computed at L1 / L3 = 1,
       1.1, 1.2 ... 10.0 and at L2 / L3 = 1.0, 1.1 ... L1 / L3. For single
       tensor LUTs, the default step is 0.2, for two-tensor LUTs it is
       0.02.
       flag: -step %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
trace: (a float)
        Trace of the diffusion tensor(s) used in the test function in the
        LUT generation. The default is 2100E-12 m^2 s^-1.
        flag: -trace %G
watson: (a boolean)
         Compute a LUT for the Watson PDF.
         flag: -watson

```

Outputs:

```

dtLUT: (an existing file name)
        Lookup Table

```

DTMetric

[Link to code](#)

Wraps command **dtshape**

Computes tensor metric statistics based on the eigenvalues $\lambda_1 \geq \lambda_2 \geq \lambda_3$ typically obtained from `ComputeEigensystem`.

The full list of statistics is:

- $\langle cl \rangle = (\lambda_1 - \lambda_2) / \lambda_1$, a measure of linearity
- $\langle cp \rangle = (\lambda_2 - \lambda_3) / \lambda_1$, a measure of planarity
- $\langle cs \rangle = \lambda_3 / \lambda_1$, a measure of isotropy with: $cl + cp + cs = 1$
- $\langle l1 \rangle$ = first eigenvalue
- $\langle l2 \rangle$ = second eigenvalue
- $\langle l3 \rangle$ = third eigenvalue
- $\langle tr \rangle = \lambda_1 + \lambda_2 + \lambda_3$
- $\langle md \rangle = tr / 3$
- $\langle rd \rangle = (\lambda_2 + \lambda_3) / 2$
- $\langle fa \rangle$ = fractional anisotropy. (Basser et al, J Magn Reson B 1996)
- $\langle ra \rangle$ = relative anisotropy (Basser et al, J Magn Reson B 1996)
- $\langle 2dfa \rangle$ = 2D FA of the two minor eigenvalues λ_2 and λ_3 i.e. $\sqrt{2 * [(\lambda_2 - \langle l \rangle)^2 + (\lambda_3 - \langle l \rangle)^2]} / (\lambda_2^2 + \lambda_3^2)$ with: $\langle l \rangle = (\lambda_2 + \lambda_3) / 2$

Example

Compute the CP planar metric as float data type.

```
>>> import nipy.interfaces.camino as cam
>>> dtmetric = cam.DTMetric()
>>> dtmetric.inputs.eigen_data = 'dteig.Bdouble'
>>> dtmetric.inputs.metric = 'cp'
>>> dtmetric.inputs.outputdatatype = 'float'
>>> dtmetric.run()
```

Inputs:

```
[Mandatory]
eigen_data: (an existing file name)
            voxel-order data filename
            flag: -inputfile %s
metric: ('fa' or 'md' or 'rd' or 'l1' or 'l2' or 'l3' or 'tr' or 'ra'
         or '2dfa' or 'cl' or 'cp' or 'cs')
         Specifies the metric to compute. Possible choices are: "fa", "md",
         "rd", "l1", "l2", "l3", "tr", "ra", "2dfa", "cl", "cp" or "cs".
         flag: -stat %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
data_header: (an existing file name)
             A Nifti .nii or .nii.gz file containing the header information.
             Usually this will be the header of the raw data file from which the
             diffusion tensors were reconstructed.
             flag: -header %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
inputdatatype: ('double' or 'float' or 'long' or 'int' or 'short' or
               'char', nipy default value: double)
    Specifies the data type of the input data. The data type can be any
    of the following strings: "char", "short", "int", "long", "float" or
    "double". Default is double data type
    flag: -inputdatatype %s
outputdatatype: ('double' or 'float' or 'long' or 'int' or 'short' or
                'char', nipy default value: double)
    Specifies the data type of the output data. The data type can be any
    of the following strings: "char", "short", "int", "long", "float" or
    "double". Default is double data type
    flag: -outputdatatype %s
outputfile: (a file name)
    Output name. Output will be a .nii.gz file if data_header is
    provided and in voxel order with outputdatatype datatype (default:
    double) otherwise.
    flag: -outputfile %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

metric_stats: (an existing file name)
    Diffusion Tensor statistics of the chosen metric

```

ModelFit[Link to code](#)**Wraps command `modelfit`**

Fits models of the spin-displacement density to diffusion MRI measurements.

This is an interface to various model fitting routines for diffusion MRI data that fit models of the spin-displacement density function. In particular, it will fit the diffusion tensor to a set of measurements as well as various other models including two or three-tensor models. The program can read input data from a file or can generate synthetic data using various test functions for testing and simulations.

Example

```

>>> import nipy.interfaces.camino as cmon
>>> fit = cmon.ModelFit()
>>> fit.model = 'dt'
>>> fit.inputs.scheme_file = 'A.scheme'
>>> fit.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> fit.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    voxel-order data filename
    flag: -inputfile %s
model: ('dt' or 'restore' or 'algdt' or 'nldt_pos' or 'nldt' or
        'ldt_wtd' or 'adc' or 'ball_stick' or 'cylcyl dt' or 'cylcyl
        restore' or 'cylcyl algdt' or 'cylcyl nldt_pos' or 'cylcyl nldt' or

```

```
'cylcyl ldt_wtd' or 'cylcyl adc' or 'cylcyl ball_stick' or
'cylcyl_eq dt' or 'cylcyl_eq restore' or 'cylcyl_eq algdt' or
'cylcyl_eq nldt_pos' or 'cylcyl_eq nldt' or 'cylcyl_eq ldt_wtd' or
'cylcyl_eq adc' or 'cylcyl_eq ball_stick' or 'pospos dt' or 'pospos
restore' or 'pospos algdt' or 'pospos nldt_pos' or 'pospos nldt' or
'pospos ldt_wtd' or 'pospos adc' or 'pospos ball_stick' or
'pospos_eq dt' or 'pospos_eq restore' or 'pospos_eq algdt' or
'pospos_eq nldt_pos' or 'pospos_eq nldt' or 'pospos_eq ldt_wtd' or
'pospos_eq adc' or 'pospos_eq ball_stick' or 'poscyl dt' or 'poscyl
restore' or 'poscyl algdt' or 'poscyl nldt_pos' or 'poscyl nldt' or
'poscyl ldt_wtd' or 'poscyl adc' or 'poscyl ball_stick' or
'poscyl_eq dt' or 'poscyl_eq restore' or 'poscyl_eq algdt' or
'poscyl_eq nldt_pos' or 'poscyl_eq nldt' or 'poscyl_eq ldt_wtd' or
'poscyl_eq adc' or 'poscyl_eq ball_stick' or 'cylcylcyl dt' or
'cylcylcyl restore' or 'cylcylcyl algdt' or 'cylcylcyl nldt_pos' or
'cylcylcyl nldt' or 'cylcylcyl ldt_wtd' or 'cylcylcyl adc' or
'cylcylcyl ball_stick' or 'cylcylcyl_eq dt' or 'cylcylcyl_eq
restore' or 'cylcylcyl_eq algdt' or 'cylcylcyl_eq nldt_pos' or
'cylcylcyl_eq nldt' or 'cylcylcyl_eq ldt_wtd' or 'cylcylcyl_eq adc'
or 'cylcylcyl_eq ball_stick' or 'pospospos dt' or 'pospospos
restore' or 'pospospos algdt' or 'pospospos nldt_pos' or 'pospospos
nldt' or 'pospospos ldt_wtd' or 'pospospos adc' or 'pospospos
ball_stick' or 'pospospos_eq dt' or 'pospospos_eq restore' or
'pospospos_eq algdt' or 'pospospos_eq nldt_pos' or 'pospospos_eq
nldt' or 'pospospos_eq ldt_wtd' or 'pospospos_eq adc' or
'pospospos_eq ball_stick' or 'posposcyl dt' or 'posposcyl restore'
or 'posposcyl algdt' or 'posposcyl nldt_pos' or 'posposcyl nldt' or
'posposcyl ldt_wtd' or 'posposcyl adc' or 'posposcyl ball_stick' or
'posposcyl_eq dt' or 'posposcyl_eq restore' or 'posposcyl_eq algdt'
or 'posposcyl_eq nldt_pos' or 'posposcyl_eq nldt' or 'posposcyl_eq
ldt_wtd' or 'posposcyl_eq adc' or 'posposcyl_eq ball_stick' or
'poscylcyl dt' or 'poscylcyl restore' or 'poscylcyl algdt' or
'poscylcyl nldt_pos' or 'poscylcyl nldt' or 'poscylcyl ldt_wtd' or
'poscylcyl adc' or 'poscylcyl ball_stick' or 'poscylcyl_eq dt' or
'poscylcyl_eq restore' or 'poscylcyl_eq algdt' or 'poscylcyl_eq
nldt_pos' or 'poscylcyl_eq nldt' or 'poscylcyl_eq ldt_wtd' or
'poscylcyl_eq adc' or 'poscylcyl_eq ball_stick')
```

Specifies the model to be fit to the data.

flag: -model %s

scheme_file: (an existing file name)

Camino scheme file (b values / vectors, see camino.fsl2scheme)

flag: -schemefile %s

[Optional]

args: (a unicode string)

Additional parameters to the command

flag: %s

bgmask: (an existing file name)

Provides the name of a file containing a background mask computed using, for example, FSL's bet2 program. The mask file contains zero in background voxels and non-zero in foreground.

flag: -bgmask %s

bgthresh: (a float)

Sets a threshold on the average $q=0$ measurement to separate foreground and background. The program does not process background voxels, but outputs the same number of values in background voxels and foreground voxels. Each value is zero in background voxels apart from the exit code which is -1.

```

    flag: -bgthresh %G
cfthresh: (a float)
    Sets a threshold on the average q=0 measurement to determine which
    voxels are CSF. This program does not treat CSF voxels any different
    to other voxels.
    flag: -csfthresh %G
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixedbvalue: (a list of from 3 to 3 items which are a float)
    As above, but specifies <M> <N> <b>. The resulting scheme is the
    same whether you specify b directly or indirectly using -fixedmodq.
    flag: -fixedbvalue %s
fixedmodq: (a list of from 4 to 4 items which are a float)
    Specifies <M> <N> <Q> <tau> a spherical acquisition scheme with M
    measurements with q=0 and N measurements with |q|=Q and diffusion
    time tau. The N measurements with |q|=Q have unique directions. The
    program reads in the directions from the files in directory
    PointSets.
    flag: -fixedmod %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputdatatype: ('float' or 'char' or 'short' or 'int' or 'long' or
    'double')
    Specifies the data type of the input file: "char", "short", "int",
    "long", "float" or "double". The input file must have BIG-ENDIAN
    ordering. By default, the input type is "float".
    flag: -inputdatatype %s
noisemap: (an existing file name)
    Specifies the name of the file to contain the estimated noise
    variance on the diffusion-weighted signal, generated by a weighted
    tensor fit. The data type of this file is big-endian double.
    flag: -noisemap %s
out_file: (a file name)
    flag: > %s, position: -1
outlier: (an existing file name)
    Specifies the name of the file to contain the outlier map generated
    by the RESTORE algorithm.
    flag: -outliermap %s
outputfile: (a file name)
    Filename of the output file.
    flag: -outputfile %s
residualmap: (an existing file name)
    Specifies the name of the file to contain the weighted residual
    errors after computing a weighted linear tensor fit. One value is
    produced per measurement, in voxel order. The data type of this file
    is big-endian double. Images of the residuals for each measurement
    can be extracted with shredder.
    flag: -residualmap %s
sigma: (a float)
    Specifies the standard deviation of the noise in the data. Required
    by the RESTORE algorithm.
    flag: -sigma %G
tau: (a float)
    Sets the diffusion time separately. This overrides the diffusion
    time specified in a scheme file or by a scheme index for both the

```

```

    acquisition scheme and in the data synthesis.
    flag: -tau %G
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

fitted_data: (an existing file name)
    output file of 4D volume in voxel order

```

PicoPDFs[Link to code](#)Wraps command **picopdfs**

Constructs a spherical PDF in each voxel for probabilistic tractography.

Example

```

>>> import nipy.interfaces.camino as cmon
>>> pdf = cmon.PicoPDFs()
>>> pdf.inputs.inputmodel = 'dt'
>>> pdf.inputs.luts = ['lut_file']
>>> pdf.inputs.in_file = 'voxel-order_data.Bfloat'
>>> pdf.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    voxel-order data filename
    flag: < %s, position: 1
luts: (a list of items which are an existing file name)
    Files containing the lookup tables. For tensor data, one lut must be
    specified for each type of inversion used in the image (one-tensor,
    two-tensor, three-tensor). For pds, the number of LUTs must match
    -numpds (it is acceptable to use the same LUT several times - see
    example, above). These LUTs may be generated with dtlutgen.
    flag: -luts %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
directmap: (a boolean)
    Only applicable when using pds as the inputmodel. Use direct mapping
    between the eigenvalues and the distribution parameters instead of
    the log of the eigenvalues.
    flag: -directmap
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run

```

```

inputmodel: ('dt' or 'multitensor' or 'pds', nipy default value:
    dt)
    input model type
    flag: -inputmodel %s, position: 2
maxcomponents: (an integer (int or long))
    The maximum number of tensor components in a voxel (default 2) for
    multitensor data. Currently, only the default is supported, but
    future releases may allow the input of three-tensor data using this
    option.
    flag: -maxcomponents %d
numpds: (an integer (int or long))
    The maximum number of PDs in a voxel (default 3) for PD data. This
    option determines the size of the input and output voxels. This means
    that the data file may be large enough to accomodate three or more
    PDs, but does not mean that any of the voxels are classified as
    containing three or more PDs.
    flag: -numpds %d
out_file: (a file name)
    flag: > %s, position: -1
pdf: ('bingham' or 'watson' or 'acg', nipy default value: bingham)
    Specifies the PDF to use. There are three choices: watson - The
    Watson distribution. This distribution is rotationally
    symmetric. bingham - The Bingham distribution, which allows
    elliptical probability density contours. acg - The Angular Central
    Gaussian distribution, which also allows elliptical probability
    density contours
    flag: -pdf %s, position: 4
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

pdfs: (an existing file name)
    path/name of 4D volume in voxel order

```

Track[Link to code](#)**Wraps command track**

Performs tractography using one of the following models: dt, 'multitensor', 'pds', 'pico', 'bootstrap', 'ball-stick', 'bayesdirac'

Example

```

>>> import nipy.interfaces.camino as cmon
>>> track = cmon.Track()
>>> track.inputs.inputmodel = 'dt'
>>> track.inputs.in_file = 'data.Bfloat'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()

```

Inputs:

```
[Mandatory]
```



```

[Optional]
anisfile: (an existing file name)
    File containing the anisotropy map. This is required to apply an
    anisotropy threshold with non tensor data. If the map is supplied it
    is always used, even in tensor data.
    flag: -anisfile %s
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    flag: -anisthresh %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
curveinterval: (a float)
    Interval over which the curvature threshold should be evaluated, in
    mm. The default is 5mm. When using the default curvature threshold
    of 90 degrees, this means that streamlines will terminate if they
    curve by more than 90 degrees over a path length of 5mm.
    flag: -curveinterval %f
    requires: curvethresh
curvethresh: (a float)
    Curvature threshold for tracking, expressed as the maximum angle (in
    degrees) between between two streamline orientations calculated over
    the length of a voxel. If the angle is greater than this, then the
    streamline terminates.
    flag: -curvethresh %f
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    flag: -datadims %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyype default value: {})
    Environment variables
gzip: (a boolean)
    save the output image in gzip format
    flag: -gzip
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    input data file
    flag: -inputfile %s, position: 1
inputdatatype: ('float' or 'double')
    input file type
    flag: -inputdatatype %s
inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or
    'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or
    'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx',
    nipyype default value: dt)
    input model type
    flag: -inputmodel %s
interpolator: ('nn' or 'prob_nn' or 'linear')
    The interpolation algorithm determines how the fiber orientation(s)
    are defined at a given continuous point within the input image.
    Interpolators are only used when the tracking algorithm is not FACT.
    The choices are: - NN: Nearest-neighbour interpolation, just uses
    the local voxel data directly.- PROB_NN: Probabilistic nearest-

```

```

neighbor interpolation, similar to the method proposed by Behrens
et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data
is not interpolated, but at each point we randomly choose one of the
8 voxels surrounding a point. The probability of choosing a
particular voxel is based on how close the point is to the centre of
that voxel.- LINEAR: Linear interpolation of the vector field
containing the principal directions at each point.
flag: -interpolator %s
ipthresh: (a float)
Curvature threshold for tracking, expressed as the minimum dot
product between two streamline orientations calculated over the
length of a voxel. If the dot product between the previous and
current directions is less than this threshold, then the streamline
terminates. The default setting will terminate fibres that curve by
more than 80 degrees. Set this to -1.0 to disable curvature checking
completely.
flag: -ipthresh %f
maxcomponents: (an integer (int or long))
The maximum number of tensor components in a voxel. This determines
the size of the input file and does not say anything about the voxel
classification. The default is 2 if the input model is multitensor
and 1 if the input model is dt.
flag: -maxcomponents %d
numpds: (an integer (int or long))
The maximum number of PDs in a voxel for input models sfpeak and
pico. The default is 3 for input model sfpeak and 1 for input model
pico. This option determines the size of the voxels in the input
file and does not affect tracking. For tensor data, use the
-maxcomponents option.
flag: -numpds %d
out_file: (a file name)
output data file
flag: -outputfile %s, position: -1
output_root: (a file name)
root directory for output
flag: -outputroot %s, position: -1
outputtracts: ('float' or 'double' or 'oogl')
output tract file type
flag: -outputtracts %s
seed_file: (an existing file name)
seed file
flag: -seedfile %s, position: 2
stepsize: (a float)
Step size for EULER and RK4 tracking. The default is 1mm.
flag: -stepsize %f
requires: tracker
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
tracker: ('fact' or 'euler' or 'rk4', nipy default value: fact)
The tracking algorithm controls streamlines are generated from the
data. The choices are: - FACT, which follows the local fibre
orientation in each voxel. No interpolation is used.- EULER, which
uses a fixed step size along the local fibre orientation. With
nearest-neighbour interpolation, this method may be very similar to
FACT, except that the step size is fixed, whereas FACT steps extend
to the boundary of the next voxel (distance variable depending on

```

```

the entry and exit points to the voxel).- RK4: Fourth-order Runge-
Kutta method. The step size is fixed, however the eventual direction
of the step is determined by taking and averaging a series of
partial steps.
flag: -tracker %s
voxel_dims: (a list of from 3 to 3 items which are a float)
voxel dimensions in mm
flag: -voxeldims %s

```

Outputs:

```

tracked: (an existing file name)
output file containing reconstructed tracts

```

TrackBallStick[Link to code](#)Wraps command **track**

Performs streamline tractography using ball-stick fitted data

Example

```

>>> import nipy.interfaces.camino as cmon
>>> track = cmon.TrackBallStick()
>>> track.inputs.in_file = 'ballstickfit_data.Bfloat'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()

```

Inputs:

```

[Mandatory]

[Optional]
anisfile: (an existing file name)
    File containing the anisotropy map. This is required to apply an
    anisotropy threshold with non tensor data. If the map is supplied it
    is always used, even in tensor data.
    flag: -anisfile %s
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    flag: -anisthresh %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
curveinterval: (a float)
    Interval over which the curvature threshold should be evaluated, in
    mm. The default is 5mm. When using the default curvature threshold
    of 90 degrees, this means that streamlines will terminate if they
    curve by more than 90 degrees over a path length of 5mm.
    flag: -curveinterval %f
    requires: curvethresh
curvethresh: (a float)
    Curvature threshold for tracking, expressed as the maximum angle (in
    degrees) between between two streamline orientations calculated over
    the length of a voxel. If the angle is greater than this, then the
    streamline terminates.

```

```

    flag: -curvethresh %f
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    flag: -datadims %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gzip: (a boolean)
    save the output image in gzip format
    flag: -gzip
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    input data file
    flag: -inputfile %s, position: 1
inputdatatype: ('float' or 'double')
    input file type
    flag: -inputdatatype %s
inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or
    'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or
    'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx',
    nipy default value: dt)
    input model type
    flag: -inputmodel %s
interpolator: ('nn' or 'prob_nn' or 'linear')
    The interpolation algorithm determines how the fiber orientation(s)
    are defined at a given continuous point within the input image.
    Interpolators are only used when the tracking algorithm is not FACT.
    The choices are: - NN: Nearest-neighbour interpolation, just uses
    the local voxel data directly.- PROB_NN: Probabilistic nearest-
    neighbor interpolation, similar to the method pro- posed by Behrens
    et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data
    is not interpolated, but at each point we randomly choose one of the
    8 voxels sur- rounding a point. The probability of choosing a
    particular voxel is based on how close the point is to the centre of
    that voxel.- LINEAR: Linear interpolation of the vector field
    containing the principal directions at each point.
    flag: -interpolator %s
ipthresh: (a float)
    Curvature threshold for tracking, expressed as the minimum dot
    product between two streamline orientations calculated over the
    length of a voxel. If the dot product between the previous and
    current directions is less than this threshold, then the streamline
    terminates. The default setting will terminate fibres that curve by
    more than 80 degrees. Set this to -1.0 to disable curvature checking
    completely.
    flag: -ipthresh %f
maxcomponents: (an integer (int or long))
    The maximum number of tensor components in a voxel. This determines
    the size of the input file and does not say anything about the voxel
    classification. The default is 2 if the input model is multitensor
    and 1 if the input model is dt.
    flag: -maxcomponents %d
numpds: (an integer (int or long))
    The maximum number of PDs in a voxel for input models sfpeak and

```

```

    pico. The default is 3 for input model sfpeak and 1 for input model
    pico. This option determines the size of the voxels in the input
    file and does not affect tracking. For tensor data, use the
    -maxcomponents option.
    flag: -numpds %d
out_file: (a file name)
    output data file
    flag: -outputfile %s, position: -1
output_root: (a file name)
    root directory for output
    flag: -outputroot %s, position: -1
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    flag: -outputtracts %s
seed_file: (an existing file name)
    seed file
    flag: -seedfile %s, position: 2
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    flag: -stepsize %f
    requires: tracker
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tracker: ('fact' or 'euler' or 'rk4', nipy default value: fact)
    The tracking algorithm controls streamlines are generated from the
    data. The choices are: - FACT, which follows the local fibre
    orientation in each voxel. No interpolation is used.- EULER, which
    uses a fixed step size along the local fibre orientation. With
    nearest-neighbour interpolation, this method may be very similar to
    FACT, except that the step size is fixed, whereas FACT steps extend
    to the boundary of the next voxel (distance variable depending on
    the entry and exit points to the voxel).- RK4: Fourth-order Runge-
    Kutta method. The step size is fixed, however the eventual direction
    of the step is determined by taking and averaging a series of
    partial steps.
    flag: -tracker %s
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    flag: -voxeldims %s

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

TrackBayesDirac[Link to code](#)Wraps command **track**

Performs streamline tractography using a Bayesian tracking with Dirac priors

Example

```

>>> import nipy.interfaces.camino as cmon
>>> track = cmon.TrackBayesDirac()

```

```
>>> track.inputs.in_file = 'tensor_fitted_data.Bdouble'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.inputs.scheme_file = 'bvecs.scheme'
>>> track.run()
```

Inputs:

```
[Mandatory]
scheme_file: (an existing file name)
    The scheme file corresponding to the data being processed.
    flag: -schemefile %s

[Optional]
anisfile: (an existing file name)
    File containing the anisotropy map. This is required to apply an
    anisotropy threshold with non tensor data. If the map is supplied it
    is always used, even in tensor data.
    flag: -anisfile %s
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    flag: -anisthresh %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
curveinterval: (a float)
    Interval over which the curvature threshold should be evaluated, in
    mm. The default is 5mm. When using the default curvature threshold
    of 90 degrees, this means that streamlines will terminate if they
    curve by more than 90 degrees over a path length of 5mm.
    flag: -curveinterval %f
    requires: curvethresh
curvepriorg: (a float)
    Concentration parameter for the prior distribution on fibre
    orientations given the fibre orientation at the previous step.
    Larger values of g make curvature less likely.
    flag: -curvepriorg %G
curvepriork: (a float)
    Concentration parameter for the prior distribution on fibre
    orientations given the fibre orientation at the previous step.
    Larger values of k make curvature less likely.
    flag: -curvepriork %G
curvethresh: (a float)
    Curvature threshold for tracking, expressed as the maximum angle (in
    degrees) between two streamline orientations calculated over
    the length of a voxel. If the angle is greater than this, then the
    streamline terminates.
    flag: -curvethresh %f
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    flag: -datadims %s
datamodel: ('cylsymmdt' or 'ballstick')
    Model of the data for Bayesian tracking. The default model is
    "cylsymmdt", a diffusion tensor with cylindrical symmetry about e_1,
    ie L1 >= L2 = L3. The other model is "ballstick", the partial
    volume model (see ballstickfit).
    flag: -datamodel %s
```

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
extpriordatatype: ('float' or 'double')
         Datatype of the prior image. The default is "double".
         flag: -extpriordatatype %s
extpriorfile: (an existing file name)
         Path to a PICO image produced by picopdfs. The PDF in each voxel is
         used as a prior for the fibre orientation in Bayesian tracking. The
         prior image must be in the same space as the diffusion data.
         flag: -extpriorfile %s
gzip: (a boolean)
         save the output image in gzip format
         flag: -gzip
ignore_exception: (a boolean, nipy default value: False)
         Print an error message instead of throwing an exception in case the
         interface fails to run
in_file: (an existing file name)
         input data file
         flag: -inputfile %s, position: 1
inputdatatype: ('float' or 'double')
         input file type
         flag: -inputdatatype %s
inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or
             'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or
             'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx',
             nipy default value: dt)
         input model type
         flag: -inputmodel %s
interpolator: ('nn' or 'prob_nn' or 'linear')
         The interpolation algorithm determines how the fiber orientation(s)
         are defined at a given continuous point within the input image.
         Interpolators are only used when the tracking algorithm is not FACT.
         The choices are: - NN: Nearest-neighbour interpolation, just uses
         the local voxel data directly.- PROB_NN: Probabilistic nearest-
         neighbor interpolation, similar to the method proposed by Behrens
         et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data
         is not interpolated, but at each point we randomly choose one of the
         8 voxels surrounding a point. The probability of choosing a
         particular voxel is based on how close the point is to the centre of
         that voxel.- LINEAR: Linear interpolation of the vector field
         containing the principal directions at each point.
         flag: -interpolator %s
ipthresh: (a float)
         Curvature threshold for tracking, expressed as the minimum dot
         product between two streamline orientations calculated over the
         length of a voxel. If the dot product between the previous and
         current directions is less than this threshold, then the streamline
         terminates. The default setting will terminate fibres that curve by
         more than 80 degrees. Set this to -1.0 to disable curvature checking
         completely.
         flag: -ipthresh %f
iterations: (an integer (int or long))
         Number of streamlines to generate at each seed point. The default is
         5000.
         flag: -iterations %d
maxcomponents: (an integer (int or long))

```

```

    The maximum number of tensor components in a voxel. This determines
    the size of the input file and does not say anything about the voxel
    classification. The default is 2 if the input model is multitensor
    and 1 if the input model is dt.
    flag: -maxcomponents %d
numpds: (an integer (int or long))
    The maximum number of PDs in a voxel for input models sfpeak and
    pico. The default is 3 for input model sfpeak and 1 for input model
    pico. This option determines the size of the voxels in the input
    file and does not affect tracking. For tensor data, use the
    -maxcomponents option.
    flag: -numpds %d
out_file: (a file name)
    output data file
    flag: -outputfile %s, position: -1
output_root: (a file name)
    root directory for output
    flag: -outputroot %s, position: -1
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    flag: -outputtracts %s
pdf: ('bingham' or 'watson' or 'acg')
    Specifies the model for PICO priors (not the curvature priors). The
    default is "bingham".
    flag: -pdf %s
pointset: (an integer (int or long))
    Index to the point set to use for Bayesian likelihood calculation.
    The index specifies a set of evenly distributed points on the unit
    sphere, where each point x defines two possible step directions (x
    or -x) for the streamline path. A larger number indexes a larger
    point set, which gives higher angular resolution at the expense of
    computation time. The default is index 1, which gives 1922 points,
    index 0 gives 1082 points, index 2 gives 3002 points.
    flag: -pointset %s
seed_file: (an existing file name)
    seed file
    flag: -seedfile %s, position: 2
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    flag: -stepsize %f
    requires: tracker
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tracker: ('fact' or 'euler' or 'rk4', nipy default value: fact)
    The tracking algorithm controls streamlines are generated from the
    data. The choices are: - FACT, which follows the local fibre
    orientation in each voxel. No interpolation is used.- EULER, which
    uses a fixed step size along the local fibre orientation. With
    nearest-neighbour interpolation, this method may be very similar to
    FACT, except that the step size is fixed, whereas FACT steps extend
    to the boundary of the next voxel (distance variable depending on
    the entry and exit points to the voxel).- RK4: Fourth-order Runge-
    Kutta method. The step size is fixed, however the eventual direction
    of the step is determined by taking and averaging a series of
    partial steps.
    flag: -tracker %s

```



```
voxel_dims: (a list of from 3 to 3 items which are a float)
            voxel dimensions in mm
            flag: -voxeldims %s
```

Outputs:

```
tracked: (an existing file name)
         output file containing reconstructed tracts
```

TrackBedpostxDeter

[Link to code](#)

Wraps command **track**

Data from FSL's bedpostx can be imported into Camino for deterministic tracking. (Use TrackBedpostxProba for bedpostx probabilistic tractography.)

The tracking is based on the vector images dyads1.nii.gz, ..., dyadsN.nii.gz, where there are a maximum of N compartments (corresponding to each fiber population) in each voxel.

It also uses the N images mean_flsamples.nii.gz, ..., mean_fNsamples.nii.gz, normalized such that the sum of all compartments is 1. Compartments where the mean_f is less than a threshold are discarded and not used for tracking. The default value is 0.01. This can be changed with the min_vol_frac option.

Example

```
>>> import nipyype.interfaces.camino as cam
>>> track = cam.TrackBedpostxDeter()
>>> track.inputs.bedpostxdir = 'bedpostxout'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()
```

Inputs:

```
[Mandatory]
bedpostxdir: (an existing directory name)
             Directory containing bedpostx output
             flag: -bedpostxdir %s

[Optional]
anisfile: (an existing file name)
          File containing the anisotropy map. This is required to apply an
          anisotropy threshold with non tensor data. If the map is supplied it
          is always used, even in tensor data.
          flag: -anisfile %s
anisthresh: (a float)
            Terminate fibres that enter a voxel with lower anisotropy than the
            threshold.
            flag: -anisthresh %f
args: (a unicode string)
      Additional parameters to the command
      flag: %s
curveinterval: (a float)
              Interval over which the curvature threshold should be evaluated, in
              mm. The default is 5mm. When using the default curvature threshold
              of 90 degrees, this means that streamlines will terminate if they
              curve by more than 90 degrees over a path length of 5mm.
              flag: -curveinterval %f
              requires: curvethresh
curvethresh: (a float)
```

```

    Curvature threshold for tracking, expressed as the maximum angle (in
    degrees) between between two streamline orientations calculated over
    the length of a voxel. If the angle is greater than this, then the
    streamline terminates.
    flag: -curvethresh %f
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    flag: -datadims %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gzip: (a boolean)
    save the output image in gzip format
    flag: -gzip
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    input data file
    flag: -inputfile %s, position: 1
inputdatatype: ('float' or 'double')
    input file type
    flag: -inputdatatype %s
inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or
    'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or
    'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx',
    nipy default value: dt)
    input model type
    flag: -inputmodel %s
interpolator: ('nn' or 'prob_nn' or 'linear')
    The interpolation algorithm determines how the fiber orientation(s)
    are defined at a given continuous point within the input image.
    Interpolators are only used when the tracking algorithm is not FACT.
    The choices are: - NN: Nearest-neighbour interpolation, just uses
    the local voxel data directly.- PROB_NN: Probabilistic nearest-
    neighbor interpolation, similar to the method pro- posed by Behrens
    et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data
    is not interpolated, but at each point we randomly choose one of the
    8 voxels sur- rounding a point. The probability of choosing a
    particular voxel is based on how close the point is to the centre of
    that voxel.- LINEAR: Linear interpolation of the vector field
    containing the principal directions at each point.
    flag: -interpolator %s
ipthresh: (a float)
    Curvature threshold for tracking, expressed as the minimum dot
    product between two streamline orientations calculated over the
    length of a voxel. If the dot product between the previous and
    current directions is less than this threshold, then the streamline
    terminates. The default setting will terminate fibres that curve by
    more than 80 degrees. Set this to -1.0 to disable curvature checking
    completely.
    flag: -ipthresh %f
maxcomponents: (an integer (int or long))
    The maximum number of tensor components in a voxel. This determines
    the size of the input file and does not say anything about the voxel
    classification. The default is 2 if the input model is multitensor

```

```

        and 1 if the input model is dt.
        flag: -maxcomponents %d
min_vol_frac: (a float)
        Zeros out compartments in bedpostx data with a mean volume fraction
        f of less than min_vol_frac. The default is 0.01.
        flag: -bedpostxminf %d
numpds: (an integer (int or long))
        The maximum number of PDs in a voxel for input models sfpeak and
        pico. The default is 3 for input model sfpeak and 1 for input model
        pico. This option determines the size of the voxels in the input
        file and does not affect tracking. For tensor data, use the
        -maxcomponents option.
        flag: -numpds %d
out_file: (a file name)
        output data file
        flag: -outputfile %s, position: -1
output_root: (a file name)
        root directory for output
        flag: -outputroot %s, position: -1
outputtracts: ('float' or 'double' or 'oogl')
        output tract file type
        flag: -outputtracts %s
seed_file: (an existing file name)
        seed file
        flag: -seedfile %s, position: 2
stepsize: (a float)
        Step size for EULER and RK4 tracking. The default is 1mm.
        flag: -stepsize %f
        requires: tracker
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
tracker: ('fact' or 'euler' or 'rk4', nipy default value: fact)
        The tracking algorithm controls streamlines are generated from the
        data. The choices are: - FACT, which follows the local fibre
        orientation in each voxel. No interpolation is used.- EULER, which
        uses a fixed step size along the local fibre orientation. With
        nearest-neighbour interpolation, this method may be very similar to
        FACT, except that the step size is fixed, whereas FACT steps extend
        to the boundary of the next voxel (distance variable depending on
        the entry and exit points to the voxel).- RK4: Fourth-order Runge-
        Kutta method. The step size is fixed, however the eventual direction
        of the step is determined by taking and averaging a series of
        partial steps.
        flag: -tracker %s
voxel_dims: (a list of from 3 to 3 items which are a float)
        voxel dimensions in mm
        flag: -voxeldims %s

```

Outputs:

```

tracked: (an existing file name)
        output file containing reconstructed tracts

```

TrackBedpostxProba[Link to code](#)

Wraps command **track**

Data from FSL's bedpostx can be imported into Camino for probabilistic tracking. (Use TrackBedpostxDeter for bedpostx deterministic tractography.)

The tracking uses the files merged_th1samples.nii.gz, merged_ph1samples.nii.gz, ..., merged_thNsamples.nii.gz, merged_phNsamples.nii.gz where there are a maximum of N compartments (corresponding to each fiber population) in each voxel. These images contain M samples of theta and phi, the polar coordinates describing the “stick” for each compartment. At each iteration, a random number X between 1 and M is drawn and the Xth samples of theta and phi become the principal directions in the voxel.

It also uses the N images mean_f1samples.nii.gz, ..., mean_fNsamples.nii.gz, normalized such that the sum of all compartments is 1. Compartments where the mean_f is less than a threshold are discarded and not used for tracking. The default value is 0.01. This can be changed with the min_vol_frac option.

Example

```
>>> import nipyne.interfaces.camino as cam
>>> track = cam.TrackBedpostxProba()
>>> track.inputs.bedpostxdir = 'bedpostxout'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.inputs.iterations = 100
>>> track.run()
```

Inputs:

```
[Mandatory]
bedpostxdir: (an existing directory name)
    Directory containing bedpostx output
    flag: -bedpostxdir %s

[Optional]
anisfile: (an existing file name)
    File containing the anisotropy map. This is required to apply an
    anisotropy threshold with non tensor data. If the map is supplied it
    is always used, even in tensor data.
    flag: -anisfile %s
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    flag: -anisthresh %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
curveinterval: (a float)
    Interval over which the curvature threshold should be evaluated, in
    mm. The default is 5mm. When using the default curvature threshold
    of 90 degrees, this means that streamlines will terminate if they
    curve by more than 90 degrees over a path length of 5mm.
    flag: -curveinterval %f
    requires: curvethresh
curvethresh: (a float)
    Curvature threshold for tracking, expressed as the maximum angle (in
    degrees) between between two streamline orientations calculated over
    the length of a voxel. If the angle is greater than this, then the
    streamline terminates.
    flag: -curvethresh %f
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
```

```

    flag: -datadims %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gzip: (a boolean)
    save the output image in gzip format
    flag: -gzip
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    input data file
    flag: -inputfile %s, position: 1
inputdatatype: ('float' or 'double')
    input file type
    flag: -inputdatatype %s
inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or
    'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or
    'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx',
    nipy default value: dt)
    input model type
    flag: -inputmodel %s
interpolator: ('nn' or 'prob_nn' or 'linear')
    The interpolation algorithm determines how the fiber orientation(s)
    are defined at a given continuous point within the input image.
    Interpolators are only used when the tracking algorithm is not FACT.
    The choices are: - NN: Nearest-neighbour interpolation, just uses
    the local voxel data directly.- PROB_NN: Probabilistic nearest-
    neighbor interpolation, similar to the method pro- posed by Behrens
    et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data
    is not interpolated, but at each point we randomly choose one of the
    8 voxels sur- rounding a point. The probability of choosing a
    particular voxel is based on how close the point is to the centre of
    that voxel.- LINEAR: Linear interpolation of the vector field
    containing the principal directions at each point.
    flag: -interpolator %s
ipthresh: (a float)
    Curvature threshold for tracking, expressed as the minimum dot
    product between two streamline orientations calculated over the
    length of a voxel. If the dot product between the previous and
    current directions is less than this threshold, then the streamline
    terminates. The default setting will terminate fibres that curve by
    more than 80 degrees. Set this to -1.0 to disable curvature checking
    completely.
    flag: -ipthresh %f
iterations: (an integer (int or long))
    Number of streamlines to generate at each seed point. The default is
    1.
    flag: -iterations %d
maxcomponents: (an integer (int or long))
    The maximum number of tensor components in a voxel. This determines
    the size of the input file and does not say anything about the voxel
    classification. The default is 2 if the input model is multitensor
    and 1 if the input model is dt.
    flag: -maxcomponents %d
min_vol_frac: (a float)
    Zeros out compartments in bedpostx data with a mean volume fraction

```

```

    f of less than min_vol_frac. The default is 0.01.
    flag: -bedpostxminf %d
numpds: (an integer (int or long))
    The maximum number of PDs in a voxel for input models sfpeak and
    pico. The default is 3 for input model sfpeak and 1 for input model
    pico. This option determines the size of the voxels in the input
    file and does not affect tracking. For tensor data, use the
    -maxcomponents option.
    flag: -numpds %d
out_file: (a file name)
    output data file
    flag: -outputfile %s, position: -1
output_root: (a file name)
    root directory for output
    flag: -outputroot %s, position: -1
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    flag: -outputtracts %s
seed_file: (an existing file name)
    seed file
    flag: -seedfile %s, position: 2
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    flag: -stepsize %f
    requires: tracker
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tracker: ('fact' or 'euler' or 'rk4', nipy default value: fact)
    The tracking algorithm controls streamlines are generated from the
    data. The choices are: - FACT, which follows the local fibre
    orientation in each voxel. No interpolation is used.- EULER, which
    uses a fixed step size along the local fibre orientation. With
    nearest-neighbour interpolation, this method may be very similar to
    FACT, except that the step size is fixed, whereas FACT steps extend
    to the boundary of the next voxel (distance variable depending on
    the entry and exit points to the voxel).- RK4: Fourth-order Runge-
    Kutta method. The step size is fixed, however the eventual direction
    of the step is determined by taking and averaging a series of
    partial steps.
    flag: -tracker %s
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    flag: -voxeldims %s

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

TrackBootstrap[Link to code](#)Wraps command **track**

Performs bootstrap streamline tractography using multiple scans of the same subject

Example

```
>>> import nipy.interfaces.camino as cmon
>>> track = cmon.TrackBootstrap()
>>> track.inputs.inputmodel='repbs_dt'
>>> track.inputs.scheme_file = 'bvecs.scheme'
>>> track.inputs.bsdatafiles = ['fitted_data1.Bfloat', 'fitted_data2.Bfloat']
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()
```

Inputs:

```
[Mandatory]
bsdatafiles: (a list of items which are an existing file name)
    Specifies files containing raw data for repetition bootstrapping.
    Use -inputfile for wild bootstrap data.
    flag: -bsdatafile %s
scheme_file: (an existing file name)
    The scheme file corresponding to the data being processed.
    flag: -schemefile %s

[Optional]
anisfile: (an existing file name)
    File containing the anisotropy map. This is required to apply an
    anisotropy threshold with non tensor data. If the map issupplied it
    is always used, even in tensor data.
    flag: -anisfile %s
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    flag: -anisthresh %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bgmask: (an existing file name)
    Provides the name of a file containing a background mask computed
    using, for example, FSL's bet2 program. The mask file contains zero
    in background voxels and non-zero in foreground.
    flag: -bgmask %s
curveinterval: (a float)
    Interval over which the curvature threshold should be evaluated, in
    mm. The default is 5mm. When using the default curvature threshold
    of 90 degrees, this means that streamlines will terminate if they
    curve by more than 90 degrees over a path length of 5mm.
    flag: -curveinterval %f
    requires: curvethresh
curvethresh: (a float)
    Curvature threshold for tracking, expressed as the maximum angle (in
    degrees) between between two streamline orientations calculated over
    the length of a voxel. If the angle is greater than this, then the
    streamline terminates.
    flag: -curvethresh %f
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    flag: -datadims %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
```

```

Environment variables
gzip: (a boolean)
    save the output image in gzip format
    flag: -gzip
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    input data file
    flag: -inputfile %s, position: 1
inputdatatype: ('float' or 'double')
    input file type
    flag: -inputdatatype %s
inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or
    'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or
    'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx',
    nipy default value: dt)
    input model type
    flag: -inputmodel %s
interpolator: ('nn' or 'prob_nn' or 'linear')
    The interpolation algorithm determines how the fiber orientation(s)
    are defined at a given continuous point within the input image.
    Interpolators are only used when the tracking algorithm is not FACT.
    The choices are: - NN: Nearest-neighbour interpolation, just uses
    the local voxel data directly.- PROB_NN: Probabilistic nearest-
    neighbor interpolation, similar to the method pro- posed by Behrens
    et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data
    is not interpolated, but at each point we randomly choose one of the
    8 voxels sur- rounding a point. The probability of choosing a
    particular voxel is based on how close the point is to the centre of
    that voxel.- LINEAR: Linear interpolation of the vector field
    containing the principal directions at each point.
    flag: -interpolator %s
inversion: (an integer (int or long))
    Tensor reconstruction algorithm for repetition bootstrapping.
    Default is 1 (linear reconstruction, single tensor).
    flag: -inversion %s
ipthresh: (a float)
    Curvature threshold for tracking, expressed as the minimum dot
    product between two streamline orientations calculated over the
    length of a voxel. If the dot product between the previous and
    current directions is less than this threshold, then the streamline
    terminates. The default setting will terminate fibres that curve by
    more than 80 degrees. Set this to -1.0 to disable curvature checking
    completely.
    flag: -ipthresh %f
iterations: (an integer (int or long))
    Number of streamlines to generate at each seed point.
    flag: -iterations %d
maxcomponents: (an integer (int or long))
    The maximum number of tensor components in a voxel. This determines
    the size of the input file and does not say anything about the voxel
    classification. The default is 2 if the input model is multitensor
    and 1 if the input model is dt.
    flag: -maxcomponents %d
numpds: (an integer (int or long))
    The maximum number of PDs in a voxel for input models sfpeak and
    pico. The default is 3 for input model sfpeak and 1 for input model

```



```

    pico. This option determines the size of the voxels in the input
    file and does not affect tracking. For tensor data, use the
    -maxcomponents option.
    flag: -numpds %d
out_file: (a file name)
    output data file
    flag: -outputfile %s, position: -1
output_root: (a file name)
    root directory for output
    flag: -outputroot %s, position: -1
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    flag: -outputtracts %s
seed_file: (an existing file name)
    seed file
    flag: -seedfile %s, position: 2
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    flag: -stepsize %f
    requires: tracker
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tracker: ('fact' or 'euler' or 'rk4', nipy default value: fact)
    The tracking algorithm controls streamlines are generated from the
    data. The choices are: - FACT, which follows the local fibre
    orientation in each voxel. No interpolation is used.- EULER, which
    uses a fixed step size along the local fibre orientation. With
    nearest-neighbour interpolation, this method may be very similar to
    FACT, except that the step size is fixed, whereas FACT steps extend
    to the boundary of the next voxel (distance variable depending on
    the entry and exit points to the voxel).- RK4: Fourth-order Runge-
    Kutta method. The step size is fixed, however the eventual direction
    of the step is determined by taking and averaging a series of
    partial steps.
    flag: -tracker %s
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    flag: -voxeldims %s

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

TrackDT[Link to code](#)**Wraps command `track`**

Performs streamline tractography using tensor data

Example

```

>>> import nipy.interfaces.camino as cmon
>>> track = cmon.TrackDT()
>>> track.inputs.in_file = 'tensor_fitted_data.Bdouble'

```

```
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()
```

Inputs:

```
[Mandatory]

[Optional]
anisfile: (an existing file name)
    File containing the anisotropy map. This is required to apply an
    anisotropy threshold with non tensor data. If the map is supplied it
    is always used, even in tensor data.
    flag: -anisfile %s
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    flag: -anisthresh %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
curveinterval: (a float)
    Interval over which the curvature threshold should be evaluated, in
    mm. The default is 5mm. When using the default curvature threshold
    of 90 degrees, this means that streamlines will terminate if they
    curve by more than 90 degrees over a path length of 5mm.
    flag: -curveinterval %f
    requires: curvethresh
curvethresh: (a float)
    Curvature threshold for tracking, expressed as the maximum angle (in
    degrees) between between two streamline orientations calculated over
    the length of a voxel. If the angle is greater than this, then the
    streamline terminates.
    flag: -curvethresh %f
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    flag: -datadims %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyype default value: {})
    Environment variables
gzip: (a boolean)
    save the output image in gzip format
    flag: -gzip
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    input data file
    flag: -inputfile %s, position: 1
inputdatatype: ('float' or 'double')
    input file type
    flag: -inputdatatype %s
inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or
    'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or
    'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx',
    nipyype default value: dt)
    input model type
```

```

    flag: -inputmodel %s
interpolator: ('nn' or 'prob_nn' or 'linear')
    The interpolation algorithm determines how the fiber orientation(s)
    are defined at a given continuous point within the input image.
    Interpolators are only used when the tracking algorithm is not FACT.
    The choices are: - NN: Nearest-neighbour interpolation, just uses
    the local voxel data directly.- PROB_NN: Probabilistic nearest-
    neighbor interpolation, similar to the method pro- posed by Behrens
    et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data
    is not interpolated, but at each point we randomly choose one of the
    8 voxels sur- rounding a point. The probability of choosing a
    particular voxel is based on how close the point is to the centre of
    that voxel.- LINEAR: Linear interpolation of the vector field
    containing the principal directions at each point.
    flag: -interpolator %s
ipthresh: (a float)
    Curvature threshold for tracking, expressed as the minimum dot
    product between two streamline orientations calculated over the
    length of a voxel. If the dot product between the previous and
    current directions is less than this threshold, then the streamline
    terminates. The default setting will terminate fibres that curve by
    more than 80 degrees. Set this to -1.0 to disable curvature checking
    completely.
    flag: -ipthresh %f
maxcomponents: (an integer (int or long))
    The maximum number of tensor components in a voxel. This determines
    the size of the input file and does not say anything about the voxel
    classification. The default is 2 if the input model is multitensor
    and 1 if the input model is dt.
    flag: -maxcomponents %d
numpds: (an integer (int or long))
    The maximum number of PDs in a voxel for input models sfpeak and
    pico. The default is 3 for input model sfpeak and 1 for input model
    pico. This option determines the size of the voxels in the input
    file and does not affect tracking. For tensor data, use the
    -maxcomponents option.
    flag: -numpds %d
out_file: (a file name)
    output data file
    flag: -outputfile %s, position: -1
output_root: (a file name)
    root directory for output
    flag: -outputroot %s, position: -1
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    flag: -outputtracts %s
seed_file: (an existing file name)
    seed file
    flag: -seedfile %s, position: 2
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    flag: -stepsize %f
    requires: tracker
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tracker: ('fact' or 'euler' or 'rk4', nipy default value: fact)

```

The tracking algorithm controls streamlines are generated from the data. The choices are: - FACT, which follows the local fibre orientation in each voxel. No interpolation is used.- EULER, which uses a fixed step size along the local fibre orientation. With nearest-neighbour interpolation, this method may be very similar to FACT, except that the step size is fixed, whereas FACT steps extend to the boundary of the next voxel (distance variable depending on the entry and exit points to the voxel).- RK4: Fourth-order Runge-Kutta method. The step size is fixed, however the eventual direction of the step is determined by taking and averaging a series of partial steps.

```

flag: -tracker %s
voxel_dims: (a list of from 3 to 3 items which are a float)
voxel dimensions in mm
flag: -voxeldims %s

```

Outputs:

```

tracked: (an existing file name)
         output file containing reconstructed tracts

```

TrackPICO[Link to code](#)Wraps command **track**

Performs streamline tractography using the Probabilistic Index of Connectivity (PICO) algorithm

Example

```

>>> import nipy.interfaces.camino as cmon
>>> track = cmon.TrackPICO()
>>> track.inputs.in_file = 'pdfs.Bfloat'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()

```

Inputs:

```

[Mandatory]

[Optional]
anisfile: (an existing file name)
          File containing the anisotropy map. This is required to apply an
          anisotropy threshold with non tensor data. If the map is supplied it
          is always used, even in tensor data.
          flag: -anisfile %s
anisthresh: (a float)
            Terminate fibres that enter a voxel with lower anisotropy than the
            threshold.
            flag: -anisthresh %f
args: (a unicode string)
      Additional parameters to the command
      flag: %s
curveinterval: (a float)
              Interval over which the curvature threshold should be evaluated, in
              mm. The default is 5mm. When using the default curvature threshold
              of 90 degrees, this means that streamlines will terminate if they
              curve by more than 90 degrees over a path length of 5mm.

```

```

        flag: -curveinterval %f
        requires: curvethresh
curvethresh: (a float)
    Curvature threshold for tracking, expressed as the maximum angle (in
    degrees) between between two streamline orientations calculated over
    the length of a voxel. If the angle is greater than this, then the
    streamline terminates.
    flag: -curvethresh %f
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    flag: -datadims %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gzip: (a boolean)
    save the output image in gzip format
    flag: -gzip
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    input data file
    flag: -inputfile %s, position: 1
inputdatatype: ('float' or 'double')
    input file type
    flag: -inputdatatype %s
inputmodel: ('dt' or 'multitensor' or 'sfpeak' or 'pico' or
    'repbs_dt' or 'repbs_multitensor' or 'ballstick' or 'wildbs_dt' or
    'bayesdirac' or 'bayesdirac_dt' or 'bedpostx_dyad' or 'bedpostx',
    nipy default value: dt)
    input model type
    flag: -inputmodel %s
interpolator: ('nn' or 'prob_nn' or 'linear')
    The interpolation algorithm determines how the fiber orientation(s)
    are defined at a given continuous point within the input image.
    Interpolators are only used when the tracking algorithm is not FACT.
    The choices are: - NN: Nearest-neighbour interpolation, just uses
    the local voxel data directly.- PROB_NN: Probabilistic nearest-
    neighbor interpolation, similar to the method pro- posed by Behrens
    et al [Magnetic Resonance in Medicine, 50:1077-1088, 2003]. The data
    is not interpolated, but at each point we randomly choose one of the
    8 voxels sur- rounding a point. The probability of choosing a
    particular voxel is based on how close the point is to the centre of
    that voxel.- LINEAR: Linear interpolation of the vector field
    containing the principal directions at each point.
    flag: -interpolator %s
ipthresh: (a float)
    Curvature threshold for tracking, expressed as the minimum dot
    product between two streamline orientations calculated over the
    length of a voxel. If the dot product between the previous and
    current directions is less than this threshold, then the streamline
    terminates. The default setting will terminate fibres that curve by
    more than 80 degrees. Set this to -1.0 to disable curvature checking
    completely.
    flag: -ipthresh %f
iterations: (an integer (int or long))

```

```

        Number of streamlines to generate at each seed point. The default is
        5000.
        flag: -iterations %d
maxcomponents: (an integer (int or long))
        The maximum number of tensor components in a voxel. This determines
        the size of the input file and does not say anything about the voxel
        classification. The default is 2 if the input model is multitensor
        and 1 if the input model is dt.
        flag: -maxcomponents %d
numpds: (an integer (int or long))
        The maximum number of PDs in a voxel for input models sfpeak and
        pico. The default is 3 for input model sfpeak and 1 for input model
        pico. This option determines the size of the voxels in the input
        file and does not affect tracking. For tensor data, use the
        -maxcomponents option.
        flag: -numpds %d
out_file: (a file name)
        output data file
        flag: -outputfile %s, position: -1
output_root: (a file name)
        root directory for output
        flag: -outputroot %s, position: -1
outputtracts: ('float' or 'double' or 'oogl')
        output tract file type
        flag: -outputtracts %s
pdf: ('bingham' or 'watson' or 'acg')
        Specifies the model for PICO parameters. The default is "bingham.
        flag: -pdf %s
seed_file: (an existing file name)
        seed file
        flag: -seedfile %s, position: 2
stepsize: (a float)
        Step size for EULER and RK4 tracking. The default is 1mm.
        flag: -stepsize %f
        requires: tracker
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
tracker: ('fact' or 'euler' or 'rk4', nipy default value: fact)
        The tracking algorithm controls streamlines are generated from the
        data. The choices are: - FACT, which follows the local fibre
        orientation in each voxel. No interpolation is used.- EULER, which
        uses a fixed step size along the local fibre orientation. With
        nearest-neighbour interpolation, this method may be very similar to
        FACT, except that the step size is fixed, whereas FACT steps extend
        to the boundary of the next voxel (distance variable depending on
        the entry and exit points to the voxel).- RK4: Fourth-order Runge-
        Kutta method. The step size is fixed, however the eventual direction
        of the step is determined by taking and averaging a series of
        partial steps.
        flag: -tracker %s
voxel_dims: (a list of from 3 to 3 items which are a float)
        voxel dimensions in mm
        flag: -voxeldims %s

```

Outputs:

```
tracked: (an existing file name)
        output file containing reconstructed tracts
```

interfaces.camino.odf

LinRecon

[Link to code](#)

Wraps command **linrecon**

Runs a linear transformation in each voxel.

Reads a linear transformation from the matrix file assuming the imaging scheme specified in the scheme file. Performs the linear transformation on the data in every voxel and outputs the result to the standard output. The output in every voxel is actually:

```
[exit code, ln(S(0)), p1, ..., pR]
```

where p1, ..., pR are the parameters of the reconstruction. Possible exit codes are:

- 0. No problems.
- 6. Bad data replaced by substitution of zero.

The matrix must be R by N+M where N+M is the number of measurements and R is the number of parameters of the reconstruction. The matrix file contains binary double-precision floats. The matrix elements are stored row by row.

Example

First run QBallMX and create a linear transform matrix using Spherical Harmonics (sh).

```
>>> import nipy.interfaces.camino as cam
>>> qballmx = cam.QBallMX()
>>> qballmx.inputs.scheme_file = 'A.scheme'
>>> qballmx.inputs.basistype = 'sh'
>>> qballmx.inputs.order = 4
>>> qballmx.run()
```

Then run it over each voxel using LinRecon

```
>>> qballcoeffs = cam.LinRecon()
>>> qballcoeffs.inputs.in_file = 'SubjectA.Bfloat'
>>> qballcoeffs.inputs.scheme_file = 'A.scheme'
>>> qballcoeffs.inputs.qball_mat = 'A_qmat.Bdouble'
>>> qballcoeffs.inputs.normalize = True
>>> qballcoeffs.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        voxel-order data filename
        flag: %s, position: 1
qball_mat: (an existing file name)
        Linear transformation matrix.
        flag: %s, position: 3
scheme_file: (an existing file name)
        Specifies the scheme file for the diffusion MRI data
        flag: %s, position: 2

[Optional]
```

```

args: (a unicode string)
    Additional parameters to the command
    flag: %s
bgmask: (an existing file name)
    background mask
    flag: -bgmask %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
log: (a boolean)
    Transform the log measurements rather than the measurements
    themselves
    flag: -log
normalize: (a boolean)
    Normalize the measurements and discard the zero measurements before
    the linear transform.
    flag: -normalize
out_file: (a file name)
    flag: > %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

recon_data: (an existing file name)
    Transformed data

```

MESD[Link to code](#)Wraps command **mesd**

MESD is a general program for maximum entropy spherical deconvolution. It also runs PASMRI, which is a special case of spherical deconvolution. The input data must be in voxel order.

The format of the output in each voxel is: { exitcode, $\ln(A^{\star}(0))$, λ_0 , λ_1 , ..., λ_N }

The exitcode contains the results of three tests. The first test thresholds the maximum relative error between the numerical integrals computed at convergence and those computed using a larger test point set; if the error is greater than a threshold the exitcode is increased from zero to one as a warning; if it is greater than a larger threshold the exitcode is increased to two to suggest failure. The second test thresholds the predicted error in numerical integrals computed using the test point set; if the predicted error is greater than a threshold the exitcode is increased by 10. The third test thresholds the RMS error between the measurements and their predictions from the fitted deconvolution; if the errors are greater than a threshold, the exit code is increased by 100. An exitcode of 112 means that all three tests were failed and the result is likely to be unreliable. If all is well the exitcode is zero. Results are often still reliable even if one or two of the tests are failed.

Other possible exitcodes are:

- 5 - The optimization failed to converge
- -1 - Background
- -100 - Something wrong in the MRI data, e.g. negative or zero measurements, so that the optimization could not run.

The standard MESD implementation is computationally demanding, particularly as the number of measurements increases (computation is approximately $O(N^2)$, where N is the number of measurements). There are two ways

to obtain significant computational speed-up:

i) Turn off error checks and use a small point set for computing numerical integrals in the algorithm by adding the flag `-fastmesd`. Sakaie CDMRI 2008 shows that using the smallest point set (`-basepointset 0`) with no error checks usually has only a minor effect on the output of the algorithm, but provides a major reduction in computation time. You can increase the point set size using `-basepointset` with an argument higher than 0, which may produce better results in some voxels, but will increase computation time, which approximately doubles every time the point set index increases by 1.

ii) Reduce the complexity of the maximum entropy encoding using `-mepointset <X>`. By default `<X> = N`, the number of measurements, and is the number of parameters in the max. ent. representation of the output function, ie the number of lambda parameters, as described in Jansons and Alexander Inverse Problems 2003. However, we can represent the function using less components and `<X>` here specifies the number of lambda parameters. To obtain speed-up, set `<X> < N`; complexity become $O(<X>^2)$ rather than $O(N^2)$. Note that `<X>` must be chosen so that the `camino/PointSets` directory contains a point set with that number of elements. When `-mepointset` decreases, the numerical integration checks make less and less of a difference and smaller point sets for numerical integration (see `-basepointset`) become adequate. So when `<X>` is low `-fastmesd` is worth using to get even more speed-up.

The choice of `<X>` is a parameter of the technique. Too low and you lose angular resolution; too high and you see no computational benefit and may even suffer from overfitting. Empirically, we have found that `<X>=16` often gives good results and good speed up, but it is worth trying a few values a comparing performance. The reduced encoding is described in the following ISMRM abstract: Sweet and Alexander “Reduced Encoding Persistent Angular Structure” 572 ISMRM 2010.

Example

Run MESD on every voxel of the data file `SubjectA.Bfloat` using the PASMRI kernel.

```
>>> import nipy.interfaces.camino as cam
>>> mesd = cam.MESD()
>>> mesd.inputs.in_file = 'SubjectA.Bfloat'
>>> mesd.inputs.scheme_file = 'A.scheme'
>>> mesd.inputs.inverter = 'PAS'
>>> mesd.inputs.inverter_param = 1.4
>>> mesd.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        voxel-order data filename
        flag: -inputfile %s, position: 1
inverter: ('SPIKE' or 'PAS')
        The inversion index specifies the type of inversion to perform on
        the data.The currently available choices are:Inverter name |
        Inverter parameters-----|-----SPIKE | bd
        (b-value x diffusivity along the fibre.)PAS | r
        flag: -filter %s, position: 2
inverter_param: (a float)
        Parameter associated with the inverter. Cf. inverter description
        formore information.
        flag: %f, position: 3
scheme_file: (an existing file name)
        Specifies the scheme file for the diffusion MRI data
        flag: -schemefile %s

[Optional]
args: (a unicode string)
        Additional parameters to the command
```

```

    flag: %s
bgmask: (an existing file name)
    background mask
    flag: -bgmask %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fastmesd: (a boolean)
    Turns off numerical integration checks and fixes the integration
    point set size at that of the index specified by -basepointset..
    flag: -fastmesd
    requires: mepointset
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputdatatype: ('float' or 'char' or 'short' or 'int' or 'long' or
    'double')
    Specifies the data type of the input file: "char", "short", "int",
    "long", "float" or "double". The input file must have BIG-ENDIAN
    ordering. By default, the input type is "float".
    flag: -inputdatatype %s
mepointset: (an integer (int or long))
    Use a set of directions other than those in the scheme file for the
    deconvolution kernel. The number refers to the number of directions
    on the unit sphere. For example, "-mepointset 54" uses the
    directions in "camino/PointSets/Elec054.txt".
    flag: -mepointset %d
out_file: (a file name)
    flag: > %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

mesd_data: (an existing file name)
    MESD data

```

QBallMX[Link to code](#)Wraps command **qballmx**

Generates a reconstruction matrix for Q-Ball. Used in LinRecon with the same scheme file to reconstruct data.

Example 1

To create a linear transform matrix using Spherical Harmonics (sh).

```

>>> import nipy.interfaces.camino as cam
>>> qballmx = cam.QBallMX()
>>> qballmx.inputs.scheme_file = 'A.scheme'
>>> qballmx.inputs.basistype = 'sh'
>>> qballmx.inputs.order = 6
>>> qballmx.run()

```

Example 2

To create a linear transform matrix using Radial Basis Functions (rbf). This command uses the default setting of rbf sigma = 0.2618 (15 degrees), data smoothing sigma = 0.1309 (7.5 degrees), rbf pointset 246

```
>>> import nipyype.interfaces.camino as cam
>>> qballmx = cam.QBallMX()
>>> qballmx.inputs.scheme_file = 'A.scheme'
>>> qballmx.run()
```

The linear transform matrix from any of these two examples can then be run over each voxel using LinRecon

```
>>> qballcoeffs = cam.LinRecon()
>>> qballcoeffs.inputs.in_file = 'SubjectA.Bfloat'
>>> qballcoeffs.inputs.scheme_file = 'A.scheme'
>>> qballcoeffs.inputs.qball_mat = 'A_qmat.Bdouble'
>>> qballcoeffs.inputs.normalize = True
>>> qballcoeffs.inputs.bgmask = 'brain_mask.nii'
>>> qballcoeffs.run()
```

Inputs:

```
[Mandatory]
scheme_file: (an existing file name)
    Specifies the scheme file for the diffusion MRI data
    flag: -schemefile %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
basistype: ('rbf' or 'sh', nipyype default value: rbf)
    Basis function type. "rbf" to use radial basis functions "sh" to use
    spherical harmonics
    flag: -basistype %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyype default value: {})
    Environment variables
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
order: (an integer (int or long))
    Specific to sh. Maximum order of the spherical harmonic series.
    Default is 4.
    flag: -order %d
out_file: (a file name)
    flag: > %s, position: -1
rbfpointset: (an integer (int or long))
    Specific to rbf. Sets the number of radial basis functions to use.
    The value specified must be present in the Pointsets directory. The
    default value is 246.
    flag: -rbfpointset %d
rbfsigma: (a float)
    Specific to rbf. Sets the width of the interpolating basis
    functions. The default value is 0.2618 (15 degrees).
    flag: -rbfsigma %f
smoothing_sigma: (a float)
    Specific to rbf. Sets the width of the smoothing basis functions.
    The default value is 0.1309 (7.5 degrees).
```

```

flag: -smoothingsigma %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

qmat: (an existing file name)
      Q-Ball reconstruction matrix

```

SFPeaks

[Link to code](#)

Wraps command **sfpeaks**

Finds the peaks of spherical functions.

This utility reads coefficients of the spherical functions and outputs a list of peak directions of the function. It computes the value of the function at each of a set of sample points. Then it finds local maxima by finding all points at which the function is larger than for any other point within a fixed search radius (the default is 0.4). The utility then uses Powell's algorithm to optimize the position of each local maximum. Finally the utility removes duplicates and tiny peaks with function value smaller than some threshold, which is the mean of the function plus some number of standard deviations. By default the program checks for consistency with a second set of starting points, but skips the optimization step. To speed up execution, you can turn off the consistency check by setting the `noconsistencycheck` flag to `True`.

By default, the utility constructs a set of sample points by randomly rotating a unit icosahedron repeatedly (the default is 1000 times, which produces a set of 6000 points) and concatenating the lists of vertices. The `'pointset = <index>'` attribute can tell the utility to use an evenly distributed set of points (index 0 gives 1082 points, 1 gives 1922, 2 gives 4322, 3 gives 8672, 4 gives 15872, 5 gives 32762, 6 gives 72032), which is quicker, because you can get away with fewer points. We estimate that you can use a factor of 2.5 less evenly distributed points than randomly distributed points and still expect similar performance levels.

The output for each voxel is:

- `exitcode` (inherited from the input data).
- `ln(A(0))`
- number of peaks found.
- flag for consistency with a repeated run (number of directions is the same and the directions are the same to within a threshold.)
- `mean(f)`.
- `std(f)`.
- direction 1 (x, y, z, f, H00, H01, H10, H11).
- direction 2 (x, y, z, f, H00, H01, H10, H11).
- direction 3 (x, y, z, f, H00, H01, H10, H11).

H is the Hessian of f at the peak. It is the matrix:

```

[d^2f/ds^2 d^2f/dsdt]
[d^2f/dtds d^2f/dt^2]
= [H00 H01]
  [H10 H11]

```

where s and t are orthogonal coordinates local to the peak.

By default the maximum number of peak directions output in each voxel is three. If less than three directions are found, zeros are output for later directions. The peaks are ordered by the value of the function at the peak. If more than the maximum number of directions are found only the strongest ones are output. The maximum number can be changed setting the `'numpds'` attribute.

The utility can read various kinds of spherical function, but must be told what kind of function is input using the `'inputmodel'` attribute. The description of the `'inputmodel'` attribute lists additional information required by

SFPeaks for each input model.

Example

First run QBallMX and create a linear transform matrix using Spherical Harmonics (sh).

```
>>> import nipy.interfaces.camino as cam
>>> sf_peaks = cam.SFPeaks()
>>> sf_peaks.inputs.in_file = 'A_recon_params.Bdouble'
>>> sf_peaks.inputs.inputmodel = 'sh'
>>> sf_peaks.inputs.order = 4
>>> sf_peaks.inputs.density = 100
>>> sf_peaks.inputs.searchradius = 1.0
>>> sf_peaks.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Voxel-order data of spherical functions
        flag: -inputfile %s
inputmodel: ('sh' or 'maxent' or 'rbf')
        Type of functions input via in_file. Currently supported options
        are: sh - Spherical harmonic series. Specify the maximum order of
        the SH series with the "order" attribute if different from the
        default of 4. maxent - Maximum entropy representations output by
        MESD. The reconstruction directions input to MESD must be specified.
        By default this is the same set of gradient directions (excluding
        zero gradients) in the scheme file, so specify the "scheme"
        attribute unless the "mepointset" attribute was set in MESD. rbf -
        Sums of radial basis functions. Specify the pointset with the
        attribute "rbfpointset" if different from the default. See QBallMX.
        flag: -inputmodel %s

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
density: (an integer (int or long))
        The number of randomly rotated icosahedra to use in constructing the
        set of points for random sampling in the peak finding algorithm.
        Default is 1000, which works well for very spiky maxent functions.
        For other types of function, it is reasonable to set the density
        much lower and increase the search radius slightly, which speeds up
        the computation.
        flag: -density %d
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
mepointset: (an integer (int or long))
        Use a set of directions other than those in the scheme file for the
        deconvolution kernel. The number refers to the number of directions
        on the unit sphere. For example, "mepointset = 54" uses the
        directions in "camino/PointSets/Elec054.txt" Use this option only if
        you told MESD to use a custom set of directions with the same
```

```

    option. Otherwise, specify the scheme file with the "scheme"
    attribute.
    flag: -mepointset %d
noconsistencycheck: (a boolean)
    Turns off the consistency check. The output shows all consistencies
    as true.
    flag: -noconsistencycheck
numpds: (an integer (int or long))
    The largest number of peak directions to output in each voxel.
    flag: -numpds %d
order: (an integer (int or long))
    Specific to sh. Maximum order of the spherical harmonic series.
    flag: -order %d
out_file: (a file name)
    flag: > %s, position: -1
pdthresh: (a float)
    Base threshold on the actual peak direction strength divided by the
    mean of the function. The default is 1.0 (the peak must be equal or
    greater than the mean).
    flag: -pdthresh %f
pointset: (an integer (int or long))
    To sample using an evenly distributed set of points instead. The
    integer can be 0, 1, ..., 7. Index 0 gives 1082 points, 1 gives
    1922, 2 gives 3002, 3 gives 4322, 4 gives 5882, 5 gives 8672, 6
    gives 12002, 7 gives 15872.
    flag: -pointset %d
rbfpointset: (an integer (int or long))
    Specific to rbf. Sets the number of radial basis functions to use.
    The value specified must be present in the Pointsets directory. The
    default value is 246.
    flag: -rbfpointset %d
scheme_file: (an existing file name)
    Specific to maxent. Specifies the scheme file.
    flag: %s
searchradius: (a float)
    The search radius in the peak finding algorithm. The default is 0.4
    (cf. "density")
    flag: -searchradius %f
stdsfrommean: (a float)
    This is the number of standard deviations of the function to be
    added to the "pdthresh" attribute in the peak directions pruning.
    flag: -stdsfrommean %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

peaks: (an existing file name)
    Peaks of the spherical functions.

```

interfaces.camino.utils**ImageStats**[Link to code](#)

Wraps command `imagestats`

This program computes voxelwise statistics on a series of 3D images. The images must be in the same space; the operation is performed voxelwise and one output is produced per voxel.

Examples

```
>>> import nipy.interfaces.camino as cam
>>> imstats = cam.ImageStats()
>>> imstats.inputs.in_files = ['im1.nii', 'im2.nii', 'im3.nii']
>>> imstats.inputs.stat = 'max'
>>> imstats.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
    List of images to process. They must be in the same space and have
    the same dimensions.
    flag: -images %s, position: -1
output_root: (a file name)
    Filename root prepended onto the names of the output files. The
    extension will be determined from the input.
    flag: -outputroot %s
stat: ('min' or 'max' or 'mean' or 'median' or 'sum' or 'std' or
    'var')
    The statistic to compute.
    flag: -stat %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_type: ('float' or 'char' or 'short' or 'int' or 'long' or
    'double', nipy default value: float)
    A Camino data type string, default is "float". Type must be signed.
    flag: -outputdatatype %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    Path of the file computed with the statistic chosen
```


interfaces.camino2trackvis.convert

Camino2Trackvis

[Link to code](#)

Wraps command **camino_to_trackvis**

Wraps **camino_to_trackvis** from Camino-Trackvis

Convert files from camino .Bfloat format to trackvis .trk format.

Example

```
>>> import nipy.interfaces.camino2trackvis as cam2trk
>>> c2t = cam2trk.Camino2Trackvis()
>>> c2t.inputs.in_file = 'data.Bfloat'
>>> c2t.inputs.out_file = 'streamlines.trk'
>>> c2t.inputs.min_length = 30
>>> c2t.inputs.data_dims = [128, 104, 64]
>>> c2t.inputs.voxel_dims = [2.0, 2.0, 2.0]
>>> c2t.inputs.voxel_order = 'LAS'
>>> c2t.run()
```

Inputs:

```
[Mandatory]
data_dims: (a list of from 3 to 3 items which are an integer (int or
            long))
            Three comma-separated integers giving the number of voxels along
            each dimension of the source scans.
            flag: -d %s, position: 4
in_file: (an existing file name)
            The input .Bfloat (camino) file.
            flag: -i %s, position: 1
voxel_dims: (a list of from 3 to 3 items which are a float)
            Three comma-separated numbers giving the size of each voxel in mm.
            flag: -x %s, position: 5
voxel_order: (a file name)
```

```

    Set the order in which various directions were stored. Specify with
    three letters consisting of one each from the pairs LR, AP, and SI.
    These stand for Left-Right, Anterior-Posterior, and Superior-
    Inferior. Whichever is specified in each position will be the
    direction of increasing order. Read coordinate system from a NIfTI
    file.
    flag: --voxel-order %s, position: 6

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
min_length: (a float)
    The minimum length of tracts to output
    flag: -l %d, position: 3
nifti_file: (an existing file name)
    Read coordinate system from a NIfTI file.
    flag: --nifti %s, position: 7
out_file: (a file name)
    The filename to which to write the .trk (trackvis) file.
    flag: -o %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

trackvis: (an existing file name)
    The filename to which to write the .trk (trackvis) file.

```

Trackvis2Camino[Link to code](#)Wraps command **trackvis_to_camino****Inputs:**

```

[Mandatory]
in_file: (an existing file name)
    The input .trk (trackvis) file.
    flag: -i %s, position: 1

[Optional]
append_file: (an existing file name)
    A file to which the append the .Bfloat data.
    flag: -a %s, position: 2
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value

```

```
    of class 'str', nipy default value: {})  
    Environment variables  
ignore_exception: (a boolean, nipy default value: False)  
    Print an error message instead of throwing an exception in case the  
    interface fails to run  
out_file: (a file name)  
    The filename to which to write the .Bfloat (camino).  
    flag: -o %s, position: 2  
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')  
    Control terminal output: `stream` - displays to terminal immediately  
    (default), `allatonce` - waits till command is finished to display  
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
camino: (an existing file name)  
    The filename to which to write the .Bfloat (camino).
```


interfaces.cmtk.cmtk

CreateMatrix

[Link to code](#)

Performs connectivity mapping and outputs the result as a NetworkX graph and a Matlab matrix

Example

```
>>> import nipype.interfaces.cmtk as cmtk
>>> conmap = cmtk.CreateMatrix()
>>> conmap.roi_file = 'fsLUT_aparc+aseg.nii'
>>> conmap.tract_file = 'fibers.trk'
>>> conmap.run()
```

Inputs:

```
[Mandatory]
resolution_network_file: (an existing file name)
    Parcellation files from Connectome Mapping Toolkit
roi_file: (an existing file name)
    Freesurfer aparc+aseg file
tract_file: (an existing file name)
    Trackvis tract file

[Optional]
count_region_intersections: (a boolean, nipype default value: False)
    Counts all of the fiber-region traversals in the connectivity matrix
    (requires significantly more computational time)
out_endpoint_array_name: (a file name)
    Name for the generated endpoint arrays
out_fiber_length_std_matrix_mat_file: (a file name)
    Matlab matrix describing the deviation in fiber lengths connecting
    each node.
out_intersection_matrix_mat_file: (a file name)
    Matlab connectivity matrix if all region/fiber intersections are
    counted.
```

```

out_matrix_file: (a file name)
    NetworkX graph describing the connectivity
out_matrix_mat_file: (a file name, nipy default value: cmatrix.mat)
    Matlab matrix describing the connectivity
out_mean_fiber_length_matrix_mat_file: (a file name)
    Matlab matrix describing the mean fiber lengths between each node.
out_median_fiber_length_matrix_mat_file: (a file name)
    Matlab matrix describing the mean fiber lengths between each node.

```

Outputs:

```

endpoint_file: (an existing file name)
    Saved Numpy array with the endpoints of each fiber
endpoint_file_mm: (an existing file name)
    Saved Numpy array with the endpoints of each fiber (in millimeters)
fiber_label_file: (an existing file name)
    Saved Numpy array with the labels for each fiber
fiber_labels_noorphans: (an existing file name)
    Saved Numpy array with the labels for each non-orphan fiber
fiber_length_file: (an existing file name)
    Saved Numpy array with the lengths of each fiber
fiber_length_std_matrix_mat_file: (an existing file name)
    Matlab matrix describing the deviation in fiber lengths connecting
    each node.
filtered_tractographies: (a list of items which are an existing file
    name)
filtered_tractography: (an existing file name)
    TrackVis file containing only those fibers originate in one and
    terminate in another region
filtered_tractography_by_intersections: (an existing file name)
    TrackVis file containing all fibers which connect two regions
intersection_matrix_file: (an existing file name)
    NetworkX graph describing the connectivity
intersection_matrix_mat_file: (an existing file name)
    Matlab matrix describing the mean fiber lengths between each node.
matlab_matrix_files: (a list of items which are an existing file
    name)
matrix_file: (an existing file name)
    NetworkX graph describing the connectivity
matrix_files: (a list of items which are an existing file name)
matrix_mat_file: (an existing file name)
    Matlab matrix describing the connectivity
mean_fiber_length_matrix_mat_file: (an existing file name)
    Matlab matrix describing the mean fiber lengths between each node.
median_fiber_length_matrix_mat_file: (an existing file name)
    Matlab matrix describing the median fiber lengths between each node.
stats_file: (an existing file name)
    Saved Matlab .mat file with the number of fibers saved at each stage

```

CreateNodes[Link to code](#)

Generates a NetworkX graph containing nodes at the centroid of each region in the input ROI file. Node data is added from the resolution network file.

Example

```
>>> import nipy.interfaces.cmtk as cmtk
>>> mknode = cmtk.CreateNodes()
>>> mknode.inputs.roi_file = 'ROI_scale500.nii.gz'
>>> mknode.run()
```

Inputs:

```
[Mandatory]
resolution_network_file: (an existing file name)
    Parcellation file from Connectome Mapping Toolkit
roi_file: (an existing file name)
    Region of interest file

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_filename: (a file name, nipy default value: nodenetwork.pck)
    Output gpickled network with the nodes defined.
```

Outputs:

```
node_network: (a file name)
    Output gpickled network with the nodes defined.
```

ROIGen

[Link to code](#)

Generates a ROI file for connectivity mapping and a dictionary file containing relevant node information

Example

```
>>> import nipy.interfaces.cmtk as cmtk
>>> rg = cmtk.ROIgen()
>>> rg.inputs.aparc_aseg_file = 'aparc+aseg.nii'
>>> rg.inputs.use_freesurfer_LUT = True
>>> rg.inputs.freesurfer_dir = '/usr/local/freesurfer'
>>> rg.run()
```

The label dictionary is written to disk using Pickle. Resulting data can be loaded using:

```
>>> file = open("FreeSurferColorLUT_adapted_aparc+aseg_out.pck", "r")
>>> file = open("fsLUT_aparc+aseg.pck", "r")
>>> labelDict = pickle.load(file)
>>> labelDict
```

Inputs:

```
[Mandatory]
aparc_aseg_file: (an existing file name)
    Freesurfer aparc+aseg file

[Optional]
LUT_file: (an existing file name)
    Custom lookup table (cf. FreeSurferColorLUT.txt)
mutually_exclusive: use_freesurfer_LUT
```

```
freesurfer_dir: (a directory name)
    Freesurfer main directory
    requires: use_freesurfer_LUT
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_dict_file: (a file name)
    Label dictionary saved in Pickle format
out_roi_file: (a file name)
    Region of Interest file for connectivity mapping
use_freesurfer_LUT: (a boolean)
    Boolean value; Set to True to use default Freesurfer LUT, False for
    custom LUT
mutually_exclusive: LUT_file
```

Outputs:

```
dict_file: (a file name)
    Label dictionary saved in Pickle format
roi_file: (a file name)
    Region of Interest file for connectivity mapping
```

cmat()

[Link to code](#)

Create the connection matrix for each resolution using fibers and ROIs.

create_allpoints_cmat()

[Link to code](#)

Create the intersection arrays for each fiber

create_endpoints_array()

[Link to code](#)

Create the endpoints arrays for each fiber Parameters ~~~~~ fib: the fibers data voxelSize: 3-tuple containing the voxel size of the ROI image Returns ~~~~~ (endpoints: matrix of size [#fibers, 2, 3] containing for each fiber the index of its first and last point in the voxelSize volume endpointsmm): endpoints in milimeter coordinates

create_nodes()

[Link to code](#)

get_connectivity_matrix()

[Link to code](#)

get_rois_crossed()

[Link to code](#)

length()

[Link to code](#)

Euclidean length of track line

Parameters

xyz [array-like shape (N,3)] array representing x,y,z of N points in a track

along [bool, optional] If True, return array giving cumulative length along track, otherwise (default) return scalar giving total length.

Returns

L [scalar or array shape (N-1,)] scalar in case of *along* == False, giving total length, array if *along* == True, giving cumulative lengths.

Examples

```

>>> xyz = np.array([[1,1,1],[2,3,4],[0,0,0]])
>>> expected_lens = np.sqrt([1+2**2+3**2, 2**2+3**2+4**2])
>>> length(xyz) == expected_lens.sum()
True
>>> len_along = length(xyz, along=True)
>>> np.allclose(len_along, expected_lens.cumsum())
True
>>> length([])
~
>>> length([[1, 2, 3]])
~
>>> length([], along=True)
array([0])

```

save_fibers()

[Link to code](#)

Stores a new trackvis file fname using only given indices

interfaces.cmtk.convert

CFFConverter

[Link to code](#)

Creates a Connectome File Format (CFF) file from input networks, surfaces, volumes, tracts, etcetera....

Example

```

>>> import nipy.interfaces.cmtk as cmtk
>>> cvt = cmtk.CFFConverter()
>>> cvt.inputs.title = 'subject 1'
>>> cvt.inputs.gifti_surfaces = ['lh.pial_converted.gii', 'rh.pial_converted.gii']
>>> cvt.inputs.tract_files = ['streamlines.trk']
>>> cvt.inputs.gpickled_networks = ['network0.gpickle']
>>> cvt.run()

```

Inputs:

[Mandatory]

[Optional]

creator: (a unicode string)

```
    Creator
data_files: (a list of items which are an existing file name)
    list of external data files (i.e. Numpy, HD5, XML)
description: (a unicode string, nipyne default value: Created with
    the Nipyne CFF converter)
    Description
email: (a unicode string)
    Email address
gifti_labels: (a list of items which are an existing file name)
    list of GIFTI labels
gifti_surfaces: (a list of items which are an existing file name)
    list of GIFTI surfaces
gpickled_networks: (a list of items which are an existing file name)
    list of gpickled Networkx graphs
graphml_networks: (a list of items which are an existing file name)
    list of graphML networks
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
license: (a unicode string)
    License
nifti_volumes: (a list of items which are an existing file name)
    list of NIFTI volumes
out_file: (a file name, nipyne default value: connectome.cff)
    Output connectome file
publisher: (a unicode string)
    Publisher
references: (a unicode string)
    References
relation: (a unicode string)
    Relation
rights: (a unicode string)
    Rights
script_files: (a list of items which are an existing file name)
    list of script files to include
species: (a unicode string, nipyne default value: Homo sapiens)
    Species
timeseries_files: (a list of items which are an existing file name)
    list of HDF5 timeseries files
title: (a unicode string)
    Connectome Title
tract_files: (a list of items which are an existing file name)
    list of Trackvis fiber files
```

Outputs:

```
connectome_file: (an existing file name)
    Output connectome file
```

MergeCNetworks

[Link to code](#)

Merges networks from multiple CFF files into one new CFF file.

Example

```
>>> import nipy.interfaces.cmtk as cmtk
>>> mrg = cmtk.MergeCNetworks()
>>> mrg.inputs.in_files = ['subj1.cff', 'subj2.cff']
>>> mrg.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
          List of CFF files to extract networks from

[Optional]
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name, nipy default value:
           merged_network_connectome.cff)
           Output CFF file with all the networks added
```

Outputs:

```
connectome_file: (an existing file name)
                 Output CFF file with all the networks added
```

interfaces.cmtk.nbs

NetworkBasedStatistic

Link to code

Calculates and outputs the average network given a set of input NetworkX gpickle files

For documentation of Network-based statistic parameters:

https://github.com/LTS5/connectomeviewer/blob/master/cviewer/libs/pyconto/groupstatistics/nbs/_nbs.py

Example

```
>>> import nipy.interfaces.cmtk as cmtk
>>> nbs = cmtk.NetworkBasedStatistic()
>>> nbs.inputs.in_group1 = ['subj1.pck', 'subj2.pck']
>>> nbs.inputs.in_group2 = ['pat1.pck', 'pat2.pck']
>>> nbs.run()
```

Inputs:

```
[Mandatory]
in_group1: (a list of items which are an existing file name)
           Networks for the first group of subjects
in_group2: (a list of items which are an existing file name)
           Networks for the second group of subjects

[Optional]
edge_key: (a unicode string, nipy default value: number_of_fibers)
          Usually "number_of_fibers", "fiber_length_mean", "fiber_length_std"
          for matrices made with CMTKSometimes "weight" or "value" for
          functional networks.
```

```
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
node_position_network: (a file name)
    An optional network used to position the nodes for the output
    networks
number_of_permutations: (an integer (int or long), nipyne default
    value: 1000)
    Number of permutations to perform
out_nbs_network: (a file name)
    Output network with edges identified by the NBS
out_nbs_pval_network: (a file name)
    Output network with p-values to weight the edges identified by the
    NBS
t_tail: ('left' or 'right' or 'both', nipyne default value: left)
    Can be one of "left", "right", or "both"
threshold: (a float, nipyne default value: 3)
    T-statistic threshold
```

Outputs:

```
nbs_network: (an existing file name)
    Output network with edges identified by the NBS
nbs_pval_network: (an existing file name)
    Output network with p-values to weight the edges identified by the
    NBS
network_files: (a list of items which are an existing file name)
    Output network with edges identified by the NBS
```

ntwks_to_matrices()[Link to code](#)**interfaces.cmtk.nx****AverageNetworks**[Link to code](#)

Calculates and outputs the average network given a set of input NetworkX gpickle files

This interface will only keep an edge in the averaged network if that edge is present in at least half of the input networks.

Example

```
>>> import nipyne.interfaces.cmtk as cmtk
>>> avg = cmtk.AverageNetworks()
>>> avg.inputs.in_files = ['subj1.pck', 'subj2.pck']
>>> avg.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
    Networks for a group of subjects

[Optional]
```

```

group_id: (a unicode string, nipy default value: group1)
    ID for group
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_gexf_groupavg: (a file name)
    Average network saved as a .gexf file
out_gpickled_groupavg: (a file name)
    Average network saved as a NetworkX .pck
resolution_network_file: (an existing file name)
    Parcellation files from Connectome Mapping Toolkit. This is not
    necessary, but if included, the interface will output the
    statistical maps as networkx graphs.

```

Outputs:

```

gexf_groupavg: (a file name)
    Average network saved as a .gexf file
gpickled_groupavg: (a file name)
    Average network saved as a NetworkX .pck
matlab_groupavgs: (a list of items which are a file name)

```

NetworkXMetrics[Link to code](#)

Calculates and outputs NetworkX-based measures for an input network

Example

```

>>> import nipy.interfaces.cmtk as cmtk
>>> nxmetrics = cmtk.NetworkXMetrics()
>>> nxmetrics.inputs.in_file = 'subj1.pck'
>>> nxmetrics.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    Input network

[Optional]
compute_clique_related_measures: (a boolean, nipy default value:
    False)
    Computing clique-related measures (e.g. node clique number) can be
    very time consuming
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_edge_metrics_matlab: (a file name)
    Output edge metrics in MATLAB .mat format
out_global_metrics_matlab: (a file name)
    Output node metrics in MATLAB .mat format
out_k_core: (a file name, nipy default value: k_core)
    Computed k-core network stored as a NetworkX pickle.
out_k_crust: (a file name, nipy default value: k_crust)
    Computed k-crust network stored as a NetworkX pickle.
out_k_shell: (a file name, nipy default value: k_shell)

```

```
    Computed k-shell network stored as a NetworkX pickle.
out_node_metrics_matlab: (a file name)
    Output node metrics in MATLAB .mat format
out_pickled_extra_measures: (a file name, nipyne default value:
    extra_measures)
    Network measures for group 1 that return dictionaries stored as a
    Pickle.
treat_as_weighted_graph: (a boolean, nipyne default value: True)
    Some network metrics can be calculated while considering only a
    binarized version of the graph
```

Outputs:

```
edge_measure_networks: (a list of items which are a file name)
edge_measures_matlab: (a file name)
    Output edge metrics in MATLAB .mat format
global_measures_matlab: (a file name)
    Output global metrics in MATLAB .mat format
gpickled_network_files: (a list of items which are a file name)
k_core: (a file name)
    Computed k-core network stored as a NetworkX pickle.
k_crust: (a file name)
    Computed k-crust network stored as a NetworkX pickle.
k_networks: (a list of items which are a file name)
k_shell: (a file name)
    Computed k-shell network stored as a NetworkX pickle.
matlab_dict_measures: (a list of items which are a file name)
matlab_matrix_files: (a list of items which are a file name)
node_measure_networks: (a list of items which are a file name)
node_measures_matlab: (a file name)
    Output node metrics in MATLAB .mat format
pickled_extra_measures: (a file name)
    Network measures for the group that return dictionaries, stored as a
    Pickle.
```

add_dicts_by_key()

[Link to code](#)

Combines two dictionaries and adds the values for those keys that are shared

add_edge_data()

[Link to code](#)

add_node_data()

[Link to code](#)

average_networks()

[Link to code](#)

Sums the edges of input networks and divides by the number of networks Writes the average network as .pck and .gexf and returns the name of the written networks

compute_dict_measures()

[Link to code](#)

Returns a dictionary

`compute_edge_measures()`

[Link to code](#)

These return edge-based measures

`compute_network_measures()`

[Link to code](#)

`compute_node_measures()`

[Link to code](#)

These return node-based measures

`compute_singlevalued_measures()`

[Link to code](#)

Returns a single value per network

`fix_keys_for_gexf()`

[Link to code](#)

GEXF Networks can be read in Gephi, however, the keys for the node and edge IDs must be converted to strings

`read_unknown_ntwk()`

[Link to code](#)

`remove_all_edges()`

[Link to code](#)

interfaces.cmtk.parcellation

Parcellate

[Link to code](#)

Subdivides segmented ROI file into smaller subregions

This interface implements the same procedure as in the ConnectomeMapper's parcellation stage (`cmp/stages/parcellation/maskcreation.py`) for a single parcellation scheme (e.g. 'scale500').

Example

```
>>> import nipy.interfaces.cmtk as cmtk
>>> parcellate = cmtk.Parcellate()
>>> parcellate.inputs.freesurfer_dir = '.'
>>> parcellate.inputs.subjects_dir = '.'
>>> parcellate.inputs.subject_id = 'subj1'
>>> parcellate.inputs.dilation = True
>>> parcellate.inputs.parcellation_name = 'scale500'
>>> parcellate.run()
```

Inputs:

```
[Mandatory]
subject_id: (a string)
    Subject ID

[Optional]
dilation: (a boolean, nipy default value: False)
    Dilate cortical parcels? Useful for fMRI connectivity
freesurfer_dir: (an existing directory name)
    Freesurfer main directory
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_roi_file: (a file name)
    Region of Interest file for connectivity mapping
parcellation_name: ('scale33' or 'scale60' or 'scale125' or
    'scale250' or 'scale500', nipy default value: scale500)
subjects_dir: (an existing directory name)
    Freesurfer subjects directory
```

Outputs:

```
aseg_file: (an existing file name)
    Automated segmentation file converted from Freesurfer "subjects"
    directory
cc_unknown_file: (an existing file name)
    Image file with regions labelled as unknown cortical structures
dilated_roi_file_in_structural_space: (a file name)
    dilated ROI image resliced to the dimensions of the original
    structural image
ribbon_file: (an existing file name)
    Image file detailing the cortical ribbon
roi_file: (an existing file name)
    Region of Interest file for connectivity mapping
roi_file_in_structural_space: (an existing file name)
    ROI image resliced to the dimensions of the original structural
    image
roiv_file: (a file name)
    Region of Interest file for fMRI connectivity mapping
white_matter_mask_file: (an existing file name)
    White matter mask file
```

create_annot_label()[Link to code](#)**create_roi()**[Link to code](#)

Creates the ROI_%.nii.gz files using the given parcellation information from networks. Iteratively create volume.

create_wm_mask()[Link to code](#)

crop_and_move_datasets()[Link to code](#)**extract()**[Link to code](#)

Extract voxel neighbourhood Parameters ~~~~~ Z: the original data shape: tuple containing neighbourhood dimensions position: tuple containing central point indexes fill: value for the padding of Z Returns ~~~~~ R: the neighbourhood of the specified point in Z

interfaces.diffusion_toolkit.dti

DTIRecon

[Link to code](#)

Wraps command **dti_recon**

Use **dti_recon** to generate tensors and other maps

Inputs:

```
[Mandatory]
DWI: (an existing file name)
    Input diffusion volume
    flag: %s, position: 1
bvals: (an existing file name)
    b values file
bvecs: (an existing file name)
    b vectors file
    flag: -gm %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
b0_threshold: (a float)
    program will use b0 image with the given threshold to mask out high
    background of fa/adc maps. by default it will calculate threshold
    automatically. but if it failed, you need to set it manually.
    flag: -b0_th
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
image_orientation_vectors: (a list of from 6 to 6 items which are a
    float)
```

```

specify image orientation vectors. if just one argument given,
will treat it as filename and read the orientation vectors from
the file. if 6 arguments are given, will treat them as 6 float
numbers and construct the 1st and 2nd vector and calculate the 3rd
one automatically.
this information will be used to determine image orientation,
as well as to adjust gradient vectors with oblique angle when
flag: -iop %f
n_averages: (an integer (int or long))
    Number of averages
    flag: -nex %s
oblique_correction: (a boolean)
    when oblique angle(s) applied, some SIEMENS dti protocols do not
    adjust gradient accordingly, thus it requires adjustment for
    correct
    diffusion tensor calculation
    flag: -oc
out_prefix: (a unicode string, nipy default value: dti)
    Output file prefix
    flag: %s, position: 2
output_type: ('nii' or 'analyze' or 'nii.gz', nipy default
    value: nii)
    output file type
    flag: -ot %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

ADC: (an existing file name)
B0: (an existing file name)
FA: (an existing file name)
FA_color: (an existing file name)
L1: (an existing file name)
L2: (an existing file name)
L3: (an existing file name)
V1: (an existing file name)
V2: (an existing file name)
V3: (an existing file name)
exp: (an existing file name)
tensor: (an existing file name)

```

DTITracker[Link to code](#)Wraps command **dti_tracker****Inputs:**

```

[Mandatory]
mask1_file: (a file name)
    first mask image
    flag: -m %s, position: 2

[Optional]
angle_threshold: (a float)
    set angle threshold. default value is 35 degree

```

```

    flag: -at %f
angle_threshold_weight: (a float)
    set angle threshold weighting factor. weighting will be be applied
    on top of the angle_threshold
    flag: -atw %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
input_data_prefix: (a unicode string, nipy default value: dti)
    for internal naming use only
    flag: %s, position: 0
input_type: ('nii' or 'analyze' or 'nii' or 'nii.gz')
    input and output file type. accepted values are:
    analyze -> analyze format 7.5
    nii -> nifti format saved in seperate .hdr and .img file
    nii.gz -> nifti format with one .nii file
    default type is 'nii'
    flag: -it %s
invert_x: (a boolean)
    invert x component of the vector
    flag: -ix
invert_y: (a boolean)
    invert y component of the vector
    flag: -iy
invert_z: (a boolean)
    invert z component of the vector
    flag: -iz
mask1_threshold: (a float)
    threshold value for the first mask image, if not given, the program
    will try automatically find the threshold
mask2_file: (a file name)
    second mask image
    flag: -m2 %s, position: 4
mask2_threshold: (a float)
    threshold value for the second mask image, if not given, the program
    will try automatically find the threshold
output_file: (a file name, nipy default value: tracks.trk)
    flag: %s, position: 1
output_mask: (a file name)
    output a binary mask file in analyze format
    flag: -om %s
primary_vector: ('v2' or 'v3')
    which vector to use for fibre tracking: v2 or v3. If not set use v1
    flag: -%s
random_seed: (an integer (int or long))
    use random location in a voxel instead of the center of the voxel to
    seed. can also define number of seed per voxel. default is 1
    flag: -rseed
step_length: (a float)
    set step length, in the unit of minimum voxel size.

```

```

        default value is 0.5 for interpolated streamline method
        and 0.1 for other methods
    flag: -l %f
swap_xy: (a boolean)
    swap x & y vectors while tracking
    flag: -sxy
swap_yz: (a boolean)
    swap y & z vectors while tracking
    flag: -syz
swap_zx: (a boolean)
    swap x & z vectors while tracking
    flag: -szx
tensor_file: (an existing file name)
    reconstructed tensor file
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tracking_method: ('fact' or 'rk2' or 'tl' or 'sl')
    fact -> use FACT method for tracking. this is the default method.
    rk2 -> use 2nd order runge-kutta method for tracking.
    tl -> use tensorline method for tracking.
    sl -> use interpolated streamline method with fixed step-length
    flag: -%s

```

Outputs:

```

mask_file: (an existing file name)
track_file: (an existing file name)

```

interfaces.diffusion_toolkit.odf**HARDIMat**[Link to code](#)Wraps command **hardi_mat**Use **hardi_mat** to calculate a reconstruction matrix from a gradient table**Inputs:**

```

[Mandatory]
bvals: (an existing file name)
    b values file
bvecs: (an existing file name)
    b vectors file
    flag: %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run

```

```

image_info: (an existing file name)
    specify image information file. the image info file is generated
    from original dicom image by diff_unpack program and contains image
    orientation and other information needed for reconstruction and
    tracking. by default will look into the image folder for .info file
    flag: -info %s
image_orientation_vectors: (a list of from 6 to 6 items which are a
    float)
    specify image orientation vectors. if just one argument given,
    will treat it as filename and read the orientation vectors from
    the file. if 6 arguments are given, will treat them as 6 float
    numbers and construct the 1st and 2nd vector and calculate the 3rd
    one automatically.
    this information will be used to determine image orientation,
    as well as to adjust gradient vectors with oblique angle when
    flag: -iop %f
oblique_correction: (a boolean)
    when oblique angle(s) applied, some SIEMENS dti protocols do not
    adjust gradient accordingly, thus it requires adjustment for
    correct
    diffusion tensor calculation
    flag: -oc
odf_file: (an existing file name)
    filename that contains the reconstruction points on a HEMI-sphere.
    use the pre-set 181 points by default
    flag: -odf %s
order: (an integer (int or long))
    maximum order of spherical harmonics. must be even number. default
    is 4
    flag: -order %s
out_file: (a file name, nipy default value: recon_mat.dat)
    output matrix file
    flag: %s, position: 2
reference_file: (an existing file name)
    provide a dicom or nifti image as the reference for the program to
    figure out the image orientation information. if no such info was
    found in the given image header, the next 5 options -info, etc.,
    will be used if provided. if image orientation info can be found
    in the given reference, all other 5 image orientation options will
    be IGNORED
    flag: -ref %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    output matrix file

```

ODFRecon[Link to code](#)Wraps command **odf_recon**

Use odf_recon to generate tensors and other maps

Inputs:

```
[Mandatory]
DWI: (an existing file name)
    Input raw data
    flag: %s, position: 1
matrix: (an existing file name)
    use given file as reconstruction matrix.
    flag: -mat %s
n_b0: (an integer (int or long))
    number of b0 scans. by default the program gets this information
    from the number of directions and number of volumes in
    the raw data. useful when dealing with incomplete raw
    data set or only using part of raw data set to reconstruct
    flag: -b0 %s
n_directions: (an integer (int or long))
    Number of directions
    flag: %s, position: 2
n_output_directions: (an integer (int or long))
    Number of output directions
    flag: %s, position: 3

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dsi: (a boolean)
    indicates that the data is dsi
    flag: -dsi
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
filter: (a boolean)
    apply a filter (e.g. high pass) to the raw image
    flag: -f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
image_orientation_vectors: (a list of from 6 to 6 items which are a
    float)
    specify image orientation vectors. if just one argument given,
    will treat it as filename and read the orientation vectors from
    the file. if 6 arguments are given, will treat them as 6 float
    numbers and construct the 1st and 2nd vector and calculate the 3rd
    one automatically.
    this information will be used to determine image orientation,
    as well as to adjust gradient vectors with oblique angle when
    flag: -iop %f
oblique_correction: (a boolean)
    when oblique angle(s) applied, some SIEMENS dti protocols do not
    adjust gradient accordingly, thus it requires adjustment for
    correct
    diffusion tensor calculation
    flag: -oc
out_prefix: (a unicode string, nipy default value: odf)
    Output file prefix
    flag: %s, position: 4
output_entropy: (a boolean)
    output entropy map
```



```

    flag: -oe
output_type: ('nii' or 'analyze' or 'nii.gz', nipy default
    value: nii)
    output file type
    flag: -ot %s
sharpness: (a float)
    smooth or sharpen the raw data. factor > 0 is smoothing.
    factor < 0 is sharpening. default value is 0
    NOTE: this option applies to DSI study only
    flag: -s %f
subtract_background: (a boolean)
    subtract the background value before reconstruction
    flag: -bg
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

B0: (an existing file name)
DWI: (an existing file name)
ODF: (an existing file name)
entropy: (a file name)
max: (an existing file name)

```

ODFTracker[Link to code](#)**Wraps command `odf_tracker`**Use `odf_tracker` to generate track file**Inputs:**

```

[Mandatory]
ODF: (an existing file name)
mask1_file: (a file name)
    first mask image
    flag: -m %s, position: 2
max: (an existing file name)

[Optional]
angle_threshold: (a float)
    set angle threshold. default value is 35 degree for
    default tracking method and 25 for rk2
    flag: -at %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
disc: (a boolean)
    use disc tracking
    flag: -disc
dsi: (a boolean)
    specify the input odf data is dsi. because dsi recon uses fixed
    pre-calculated matrix, some special orientation patch needs to
    be applied to keep dti/dsi/q-ball consistent.
    flag: -dsi
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value

```

```

    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
image_orientation_vectors: (a list of from 6 to 6 items which are a
    float)
    specify image orientation vectors. if just one argument given,
    will treat it as filename and read the orientation vectors from
    the file. if 6 arguments are given, will treat them as 6 float
    numbers and construct the 1st and 2nd vector and calculate the 3rd
    one automatically.
    this information will be used to determine image orientation,
    as well as to adjust gradient vectors with oblique angle when
    flag: -iop %f
input_data_prefix: (a unicode string, nipy default value: odf)
    recon data prefix
    flag: %s, position: 0
input_output_type: ('nii' or 'analyze' or 'nii' or 'nii.gz', nipy
    default value: nii)
    input and output file type
    flag: -it %s
invert_x: (a boolean)
    invert x component of the vector
    flag: -ix
invert_y: (a boolean)
    invert y component of the vector
    flag: -iy
invert_z: (a boolean)
    invert z component of the vector
    flag: -iz
limit: (an integer (int or long))
    in some special case, such as heart data, some track may go into
    infinite circle and take long time to stop. this option allows
    setting a limit for the longest tracking steps (voxels)
    flag: -limit %d
mask1_threshold: (a float)
    threshold value for the first mask image, if not given, the program
    will try automatically find the threshold
mask2_file: (a file name)
    second mask image
    flag: -m2 %s, position: 4
mask2_threshold: (a float)
    threshold value for the second mask image, if not given, the program
    will try automatically find the threshold
out_file: (a file name, nipy default value: tracks.trk)
    output track file
    flag: %s, position: 1
random_seed: (an integer (int or long))
    use random location in a voxel instead of the center of the voxel
    to seed. can also define number of seed per voxel. default is 1
    flag: -rseed %s
runge_kutta2: (a boolean)
    use 2nd order runge-kutta method for tracking.
    default tracking method is non-interpolate streamline
    flag: -rk2
slice_order: (an integer (int or long))
    set the slice order. 1 means normal, -1 means reversed. default

```

```

        value is 1
        flag: -sorder %d
step_length: (a float)
    set step length, in the unit of minimum voxel size.
    default value is 0.1.
    flag: -l %f
swap_xy: (a boolean)
    swap x and y vectors while tracking
    flag: -sxy
swap_yz: (a boolean)
    swap y and z vectors while tracking
    flag: -syz
swap_zx: (a boolean)
    swap x and z vectors while tracking
    flag: -szx
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
voxel_order: ('RAS' or 'RPS' or 'RAI' or 'RPI' or 'LAI' or 'LAS' or
    'LPS' or 'LPI')
    specify the voxel order in RL/AP/IS (human brain) reference. must be
    3 letters with no space in between.
    for example, RAS means the voxel row is from L->R, the column
    is from P->A and the slice order is from I->S.
    by default voxel order is determined by the image orientation
    (but NOT guaranteed to be correct because of various standards).
    for example, siemens axial image is LPS, coronal image is LIP and
    sagittal image is PIL.
    this information also is NOT needed for tracking but will be saved
    in the track file and is essential for track display to map onto
    the right coordinates
    flag: -vorder %s

```

Outputs:

```

track_file: (an existing file name)
    output track file

```

interfaces.diffusion_toolkit.postproc

SplineFilter

[Link to code](#)Wraps command **spline_filter**

Smoothes TrackVis track files with a B-Spline filter.

Helps remove redundant track points and segments (thus reducing the size of the track file) and also make tracks nicely smoothed. It will NOT change the quality of the tracks or lose any original information.

Example

```

>>> import nipy.interfaces.diffusion_toolkit as dtk
>>> filt = dtk.SplineFilter()
>>> filt.inputs.track_file = 'tracks.trk'
>>> filt.inputs.step_length = 0.5
>>> filt.run()

```

Inputs:

```
[Mandatory]
step_length: (a float)
    in the unit of minimum voxel size
    flag: %f, position: 1
track_file: (an existing file name)
    file containing tracks to be filtered
    flag: %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_file: (a file name, nipy default value: spline_tracks.trk)
    target file for smoothed tracks
    flag: %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
smoothed_track_file: (an existing file name)
```

TrackMerge

[Link to code](#)

Wraps command **track_merge**

Merges several TrackVis track files into a single track file.

An id type property tag is added to each track in the newly merged file, with each unique id representing where the track was originally from. When the merged file is loaded in TrackVis, a property filter will show up in Track Property panel. Users can adjust that to distinguish and sub-group tracks by its id (origin).

Example

```
>>> import nipy.interfaces.diffusion_toolkit as dtk
>>> mrg = dtk.TrackMerge()
>>> mrg.inputs.track_files = ['track1.trk', 'track2.trk']
>>> mrg.run()
```

Inputs:

```
[Mandatory]
track_files: (a list of items which are an existing file name)
    file containing tracks to be filtered
    flag: %s..., position: 0

[Optional]
args: (a unicode string)
```

```
Additional parameters to the command
flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipyte default value: {})
         Environment variables
ignore_exception: (a boolean, nipyte default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
output_file: (a file name, nipyte default value: merged_tracks.trk)
              target file for merged tracks
              flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
track_file: (an existing file name)
```


interfaces.dipy.anisotropic_power

APMQball

[Link to code](#)

Calculates the anisotropic power map

Example

```
>>> import nipype.interfaces.dipy as dipy
>>> apm = dipy.APMQball()
>>> apm.inputs.in_file = 'diffusion.nii'
>>> apm.inputs.in_bvec = 'bvecs'
>>> apm.inputs.in_bval = 'bvals'
>>> apm.run()
```

Inputs:

```
[Mandatory]
in_bval: (an existing file name)
    input b-values table
in_bvec: (an existing file name)
    input b-vectors table
in_file: (an existing file name)
    input diffusion data

[Optional]
b0_thres: (an integer (int or long), nipype default value: 700)
    b0 threshold
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_file: (an existing file name)
    An optional brain mask
out_prefix: (a unicode string)
    output prefix for file names
```

Outputs:

```
out_file: (an existing file name)
```

interfaces.dipy.base

DipyBaseInterface

[Link to code](#)

A base interface for py:mod:*dipy* computations

Inputs:

```
[Mandatory]

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

Outputs:

```
None
```

DipyDiffusionInterface

[Link to code](#)

A base interface for py:mod:*dipy* computations

Inputs:

```
[Mandatory]
in_bval: (an existing file name)
    input b-values table
in_bvec: (an existing file name)
    input b-vectors table
in_file: (an existing file name)
    input diffusion data

[Optional]
b0_thres: (an integer (int or long), nipy default value: 700)
    b0 threshold
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_prefix: (a unicode string)
    output prefix for file names
```

Outputs:

```
None
```

interfaces.dipy.preprocess

Denoise

[Link to code](#)

An interface to denoising diffusion datasets [Coupe2008]. See http://nipy.org/dipy/examples_built/denoise_nlmeans.html#example-denoise-nlmeans.

Example

```
>>> import nipy.interfaces.dipy as dipy
>>> denoise = dipy.Denoise()
>>> denoise.inputs.in_file = 'diffusion.nii'
>>> denoise.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        The input 4D diffusion-weighted image file
noise_model: ('rician' or 'gaussian', nipy default value: rician)
             noise distribution model

[Optional]
block_radius: (an integer (int or long))
              block_radius
in_mask: (an existing file name)
         brain mask
noise_mask: (an existing file name)
            mask in which the standard deviation of noise will be computed
patch_radius: (an integer (int or long))
              patch radius
signal_mask: (an existing file name)
             mask in which the mean signal will be computed
snr: (a float)
     manually set an SNR
```

Outputs:

```
out_file: (an existing file name)
```

Resample

Link to code

An interface to reslicing diffusion datasets. See http://nipy.org/dipy/examples_built/reslice_datasets.html#example-reslice-datasets.

Example

```
>>> import nipy.interfaces.dipy as dipy
>>> reslice = dipy.Resample()
>>> reslice.inputs.in_file = 'diffusion.nii'
>>> reslice.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        The input 4D diffusion-weighted image file
interp: (an integer (int or long), nipy default value: 1)
        order of the interpolator (0 = nearest, 1 = linear, etc.)
```

```
[Optional]
vox_size: (a tuple of the form: (a float, a float, a float))
    specify the new voxel zooms. If no vox_size is set, then isotropic
    regridding will be performed, with spacing equal to the smallest
    current zoom.
```

Outputs:

```
out_file: (an existing file name)
```

nlmeans_proxy()

[Link to code](#)

Uses non-local means to denoise 4D datasets

resample_proxy()

[Link to code](#)

Performs regridding of an image to set isotropic voxel sizes using dipy.

interfaces.dipy.reconstruction

CSD

[Link to code](#)

Uses CSD [Tournier2007] to generate the fODF of DWIs. The interface uses dipy, as explained in dipy's CSD example.

Example

```
>>> from nipy.interfaces import dipy as ndp
>>> csd = ndp.CSD()
>>> csd.inputs.in_file = '4d_dwi.nii'
>>> csd.inputs.in_bval = 'bvals'
>>> csd.inputs.in_bvec = 'bvecs'
>>> res = csd.run()
```

Inputs:

```
[Mandatory]
in_bval: (an existing file name)
    input b-values table
in_bvec: (an existing file name)
    input b-vectors table
in_file: (an existing file name)
    input diffusion data

[Optional]
b0_thres: (an integer (int or long), nipy default value: 700)
    b0 threshold
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_mask: (an existing file name)
    input mask in which compute tensors
out_fods: (a file name)
```

```

    fODFs output file name
out_prefix: (a unicode string)
    output prefix for file names
response: (an existing file name)
    single fiber estimated response
save_fods: (a boolean, nipy default value: True)
    save fODFs in file
sh_order: (an integer (int or long), nipy default value: 8)
    maximal shperical harmonics order

```

Outputs:

```

model: (a file name)
    Python pickled object of the CSD model fitted.
out_fods: (a file name)
    fODFs output file name

```

EstimateResponseSH[Link to code](#)

Uses dipy to compute the single fiber response to be used in spherical deconvolution methods, in a similar way to MRTrix's command `estimate_response`.

Example

```

>>> from nipy.interfaces import dipy as ndp
>>> dti = ndp.EstimateResponseSH()
>>> dti.inputs.in_file = '4d_dwi.nii'
>>> dti.inputs.in_bval = 'bvals'
>>> dti.inputs.in_bvec = 'bvecs'
>>> dti.inputs.in_evals = 'dwi_evals.nii'
>>> res = dti.run()

```

Inputs:

```

[Mandatory]
in_bval: (an existing file name)
    input b-values table
in_bvec: (an existing file name)
    input b-vectors table
in_evals: (an existing file name)
    input eigenvalues file
in_file: (an existing file name)
    input diffusion data

[Optional]
auto: (a boolean)
    use the auto_response estimator from dipy
    mutually_exclusive: recursive
b0_thres: (an integer (int or long), nipy default value: 700)
    b0 threshold
fa_thresh: (a float, nipy default value: 0.7)
    FA threshold
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_mask: (an existing file name)

```

```

    input mask in which we find single fibers
out_mask: (a file name, nipy default value: wm_mask.nii.gz)
    computed wm mask
out_prefix: (a unicode string)
    output prefix for file names
recursive: (a boolean)
    use the recursive response estimator from dipy
    mutually_exclusive: auto
response: (a file name, nipy default value: response.txt)
    the output response file
roi_radius: (an integer (int or long), nipy default value: 10)
    ROI radius to be used in auto_response

```

Outputs:

```

out_mask: (an existing file name)
    output wm mask
response: (an existing file name)
    the response file

```

RESTORE

[Link to code](#)

Uses RESTORE [Chang2005] to perform DTI fitting with outlier detection. The interface uses dipy, as explained in dipy's [documentation](#).

Example

```

>>> from nipy.interfaces import dipy as ndp
>>> dti = ndp.RESTORE()
>>> dti.inputs.in_file = '4d_dwi.nii'
>>> dti.inputs.in_bval = 'bvals'
>>> dti.inputs.in_bvec = 'bvecs'
>>> res = dti.run()

```

Inputs:

```

[Mandatory]
in_bval: (an existing file name)
    input b-values table
in_bvec: (an existing file name)
    input b-vectors table
in_file: (an existing file name)
    input diffusion data

[Optional]
b0_thres: (an integer (int or long), nipy default value: 700)
    b0 threshold
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_mask: (an existing file name)
    input mask in which compute tensors
noise_mask: (an existing file name)
    input mask in which compute noise variance
out_prefix: (a unicode string)
    output prefix for file names

```

Outputs:

```

evals: (a file name)
        output the eigenvalues of the fitted DTI
evecs: (a file name)
        output the eigenvectors of the fitted DTI
fa: (a file name)
        output fractional anisotropy (FA) map computed from the fitted DTI
md: (a file name)
        output mean diffusivity (MD) map computed from the fitted DTI
mode: (a file name)
        output mode (MO) map computed from the fitted DTI
rd: (a file name)
        output radial diffusivity (RD) map computed from the fitted DTI
trace: (a file name)
        output the tensor trace map computed from the fitted DTI

```

interfaces.dipy.simulate**SimulateMultiTensor**

[Link to code](#)

Interface to MultiTensor model simulator in dipy http://nipy.org/dipy/examples_built/simulate_multi_tensor.html

Example

```

>>> import nipype.interfaces.dipy as dipy
>>> sim = dipy.SimulateMultiTensor()
>>> sim.inputs.in_dirs = ['fdir00.nii', 'fdir01.nii']
>>> sim.inputs.in_frac = ['ffra00.nii', 'ffra01.nii']
>>> sim.inputs.in_vfms = ['tpm_00.nii.gz', 'tpm_01.nii.gz',
...                       'tpm_02.nii.gz']
>>> sim.inputs.baseline = 'b0.nii'
>>> sim.inputs.in_bvec = 'bvecs'
>>> sim.inputs.in_bval = 'bvals'
>>> sim.run()

```

Inputs:

```

[Mandatory]
baseline: (an existing file name)
        baseline T2 signal
in_dirs: (a list of items which are an existing file name)
        list of fibers (principal directions)
in_frac: (a list of items which are an existing file name)
        volume fraction of each fiber
in_vfms: (a list of items which are an existing file name)
        volume fractions of isotropic compartments

[Optional]
bvalues: (a list of items which are an integer (int or long), nipype
        default value: [1000, 3000])
        list of b-values (when table is automatically generated)
diff_iso: (a list of items which are a float, nipype default value:
        [0.003, 0.00096, 0.00068])
        Diffusivity of isotropic compartments

```

```

diff_sf: (a tuple of the form: (a float, a float, a float), nipy
        default value: (0.0017, 0.0002, 0.0002))
        Single fiber tensor
gradients: (an existing file name)
        gradients file
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
in_bval: (an existing file name)
        input bvals file
in_bvec: (an existing file name)
        input bvecs file
in_mask: (an existing file name)
        mask to simulate data
n_proc: (an integer (int or long), nipy default value: 0)
        number of processes
num_dirs: (an integer (int or long), nipy default value: 32)
        number of gradient directions (when table is automatically
        generated)
out_bval: (a file name, nipy default value: bval.sim)
        simulated b values
out_bvec: (a file name, nipy default value: bvec.sim)
        simulated b vectors
out_file: (a file name, nipy default value: sim_dwi.nii.gz)
        output file with fractions to be simulated
out_mask: (a file name, nipy default value: sim_msk.nii.gz)
        file with the mask simulated
snr: (an integer (int or long), nipy default value: 0)
        signal-to-noise ratio (dB)

```

Outputs:

```

out_bval: (an existing file name)
        simulated b values
out_bvec: (an existing file name)
        simulated b vectors
out_file: (an existing file name)
        simulated DWIs
out_mask: (an existing file name)
        mask file

```

interfaces.dipy.tensors**DTI**[Link to code](#)

Calculates the diffusion tensor model parameters

Example

```

>>> import nipy.interfaces.dipy as dipy
>>> dti = dipy.DTI()
>>> dti.inputs.in_file = 'diffusion.nii'
>>> dti.inputs.in_bvec = 'bvecs'
>>> dti.inputs.in_bval = 'bvals'
>>> dti.run()

```

Inputs:

```
[Mandatory]
in_bval: (an existing file name)
         input b-values table
in_bvec: (an existing file name)
         input b-vectors table
in_file: (an existing file name)
         input diffusion data

[Optional]
b0_thres: (an integer (int or long), nipy default value: 700)
          b0 threshold
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask_file: (an existing file name)
            An optional white matter mask
out_prefix: (a unicode string)
            output prefix for file names
```

Outputs:

```
ad_file: (an existing file name)
fa_file: (an existing file name)
md_file: (an existing file name)
out_file: (an existing file name)
rd_file: (an existing file name)
```

TensorMode[Link to code](#)

Creates a map of the mode of the diffusion tensors given a set of diffusion-weighted images, as well as their associated b-values and b-vectors. Fits the diffusion tensors and calculates tensor mode with Dipy.

Example

```
>>> import nipy.interfaces.dipy as dipy
>>> mode = dipy.TensorMode()
>>> mode.inputs.in_file = 'diffusion.nii'
>>> mode.inputs.in_bvec = 'bvecs'
>>> mode.inputs.in_bval = 'bvals'
>>> mode.run()
```

Inputs:

```
[Mandatory]
in_bval: (an existing file name)
         input b-values table
in_bvec: (an existing file name)
         input b-vectors table
in_file: (an existing file name)
         input diffusion data

[Optional]
b0_thres: (an integer (int or long), nipy default value: 700)
          b0 threshold
ignore_exception: (a boolean, nipy default value: False)
```

```
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_file: (an existing file name)
    An optional white matter mask
out_prefix: (a unicode string)
    output prefix for file names
```

Outputs:

```
out_file: (an existing file name)
```

interfaces.dipy.tracks

StreamlineTractography

[Link to code](#)

Streamline tractography using EuDX [*Garyfallidis12*].

Example

```
>>> from nipy.interfaces import dipy as ndp
>>> track = ndp.StreamlineTractography()
>>> track.inputs.in_file = '4d_dwi.nii'
>>> track.inputs.in_model = 'model.pklz'
>>> track.inputs.tracking_mask = 'dilated_wm_mask.nii'
>>> res = track.run()
```

Inputs:

```
[Mandatory]
gfa_thresh: (a float, nipy default value: 0.2)
    GFA threshold to compute tracking mask
in_file: (an existing file name)
    input diffusion data
min_angle: (a float, nipy default value: 25.0)
    minimum separation angle
multiprocess: (a boolean, nipy default value: True)
    use multiprocessing
num_seeds: (an integer (int or long), nipy default value: 10000)
    desired number of tracks in tractography
peak_threshold: (a float, nipy default value: 0.5)
    threshold to consider peaks from model
save_seeds: (a boolean, nipy default value: False)
    save seeding voxels coordinates

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_model: (an existing file name)
    input f/d-ODF model extracted from.
in_peaks: (an existing file name)
    peaks computed from the odF
out_prefix: (a unicode string)
    output prefix for file names
seed_coord: (an existing file name)
    file containing the list of seed voxel coordinates (N,3)
```



```
seed_mask: (an existing file name)
    input mask within which perform seeding
tracking_mask: (an existing file name)
    input mask within which perform tracking
```

Outputs:

```
gfa: (a file name)
    The resulting GFA (generalized FA) computed using the peaks of the
    ODF
odf_peaks: (a file name)
    peaks computed from the odf
out_seeds: (a file name)
    file containing the (N,3) *voxel* coordinates used in seeding.
tracks: (a file name)
    TrackVis file containing extracted streamlines
```

TrackDensityMap[Link to code](#)

Creates a tract density image from a TrackVis track file using functions from dipy

Example

```
>>> import nipy.interfaces.dipy as dipy
>>> trk2tdi = dipy.TrackDensityMap()
>>> trk2tdi.inputs.in_file = 'converted.trk'
>>> trk2tdi.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    The input TrackVis track file

[Optional]
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    The size of the image in voxels.
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_filename: (a file name, nipy default value: tdi.nii)
    The output filename for the tracks in TrackVis (.trk) format
points_space: ('rasmm' or 'voxel' or None, nipy default value:
    rasmm)
    coordinates of trk file
reference: (an existing file name)
    A reference file to define RAS coordinates space
voxel_dims: (a list of from 3 to 3 items which are a float)
    The size of each voxel in mm.
```

Outputs:

```
out_file: (an existing file name)
```


interfaces.elastix.registration

AnalyzeWarp

[Link to code](#)

Wraps command **transformix -def all -jac all -jacmat all**

Use transformix to get details from the input transform (generate the corresponding deformation field, generate the determinant of the Jacobian map or the Jacobian map itself)

Example

```
>>> from nipy.interfaces.elastix import AnalyzeWarp
>>> reg = AnalyzeWarp()
>>> reg.inputs.transform_file = 'TransformParameters.0.txt'
>>> reg.cmdline
'transformix -def all -jac all -jacmat all -out ./ -tp TransformParameters.0.txt'
```

Inputs:

```
[Mandatory]
output_path: (an existing directory name, nipy default value: ./)
    output directory
    flag: -out %s
transform_file: (an existing file name)
    transform-parameter file, only 1
    flag: -tp %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
```

```
    Print an error message instead of throwing an exception in case the
    interface fails to run
num_threads: (an integer (int or long))
    set the maximum number of threads of elastix
    flag: -threads %0ld
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
disp_field: (a file name)
    displacements field
jacdet_map: (a file name)
    det(Jacobian) map
jacmat_map: (a file name)
    Jacobian matrix map
```

ApplyWarp

[Link to code](#)**Wraps command `transformix`**

Use `transformix` to apply a transform on an input image. The transform is specified in the transform-parameter file.

Example

```
>>> from nipy.interfaces.elastix import ApplyWarp
>>> reg = ApplyWarp()
>>> reg.inputs.moving_image = 'moving1.nii'
>>> reg.inputs.transform_file = 'TransformParameters.0.txt'
>>> reg.cmdline
'transformix -in moving1.nii -out ./ -tp TransformParameters.0.txt'
```

Inputs:

```
[Mandatory]
moving_image: (an existing file name)
    input image to deform
    flag: -in %s
output_path: (an existing directory name, nipy default value: ./)
    output directory
    flag: -out %s
transform_file: (an existing file name)
    transform-parameter file, only 1
    flag: -tp %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
num_threads: (an integer (int or long))
        set the maximum number of threads of elastix
        flag: -threads %0ld
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

warped_file: (a file name)
    input moving image warped to fixed image

```

PointsWarp

[Link to code](#)Wraps command **transformix**

Use transformix to apply a transform on an input point set. The transform is specified in the transform-parameter file.

Example

```

>>> from nipyte.interfaces.elastix import PointsWarp
>>> reg = PointsWarp()
>>> reg.inputs.points_file = 'surfl.vtk'
>>> reg.inputs.transform_file = 'TransformParameters.0.txt'
>>> reg.cmdline
'transformix -out ./ -def surfl.vtk -tp TransformParameters.0.txt'

```

Inputs:

```

[Mandatory]
output_path: (an existing directory name, nipyte default value: ./)
    output directory
    flag: -out %s
points_file: (an existing file name)
    input points (accepts .vtk triangular meshes).
    flag: -def %s
transform_file: (an existing file name)
    transform-parameter file, only 1
    flag: -tp %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
num_threads: (an integer (int or long))
    set the maximum number of threads of elastix

```

```
flag: -threads %0ld
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
warped_file: (a file name)
input points displaced in fixed image domain
```

Registration

[Link to code](#)

Wraps command **elastix**

Elastix nonlinear registration interface

Example

```
>>> from nipyne.interfaces.elastix import Registration
>>> reg = Registration()
>>> reg.inputs.fixed_image = 'fixed1.nii'
>>> reg.inputs.moving_image = 'moving1.nii'
>>> reg.inputs.parameters = ['elastix.txt']
>>> reg.cmdline
'elastix -f fixed1.nii -m moving1.nii -out ./ -p elastix.txt'
```

Inputs:

```
[Mandatory]
fixed_image: (an existing file name)
    fixed image
    flag: -f %s
moving_image: (an existing file name)
    moving image
    flag: -m %s
output_path: (an existing directory name, nipyne default value: ./)
    output directory
    flag: -out %s
parameters: (a list of items which are an existing file name)
    parameter file, elastix handles 1 or more -p
    flag: -p %s...

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
fixed_mask: (an existing file name)
    mask for fixed image
    flag: -fMask %s
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

```

initial_transform: (an existing file name)
    parameter file for initial transform
    flag: -t0 %s
moving_mask: (an existing file name)
    mask for moving image
    flag: -mMask %s
num_threads: (an integer (int or long))
    set the maximum number of threads of elastix
    flag: -threads %0ld
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

transform: (a list of items which are an existing file name)
    output transform
warped_file: (a file name)
    input moving image warped to fixed image
warped_files: (a list of items which are a file name)
    input moving image warped to fixed image at each level
warped_files_flags: (a list of items which are a boolean)
    flag indicating if warped image was generated

```

interfaces.elastix.utils

EditTransform

[Link to code](#)

Manipulates an existing transform file generated with elastix

Example

```

>>> from nipy.interfaces.elastix import EditTransform
>>> tfm = EditTransform()
>>> tfm.inputs.transform_file = 'TransformParameters.0.txt'
>>> tfm.inputs.reference_image = 'fixed1.nii'
>>> tfm.inputs.output_type = 'unsigned char'
>>> tfm.run()

```

Inputs:

```

[Mandatory]
transform_file: (an existing file name)
    transform-parameter file, only 1

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interpolation: ('cubic' or 'linear' or 'nearest', nipy default
    value: cubic)
    set a new interpolator for transformation
    flag: FinalBSplineInterpolationOrder
output_file: (a file name)
    the filename for the resulting transform file

```

```
output_format: ('nii.gz' or 'nii' or 'mhd' or 'hdr' or 'vtk')
    set a new image format for resampled images
    flag: ResultImageFormat
output_type: ('float' or 'unsigned char' or 'unsigned short' or
    'short' or 'unsigned long' or 'long' or 'double')
    set a new output pixel type for resampled images
    flag: ResultImagePixelType
reference_image: (an existing file name)
    set a new reference image to change the target coordinate system.
```

Outputs:

```
output_file: (an existing file name)
    output transform file
```


interfaces.freesurfer.longitudinal

FuseSegmentations

[Link to code](#)

Wraps command **mri_fuse_segmentations**

fuse segmentations together from multiple timepoints

Examples

```
>>> from nipy.interfaces.freesurfer import FuseSegmentations
>>> fuse = FuseSegmentations()
>>> fuse.inputs.subject_id = 'tp.long.A.template'
>>> fuse.inputs.timepoints = ['tp1', 'tp2']
>>> fuse.inputs.out_file = 'aseg.fused.mgz'
>>> fuse.inputs.in_segmentations = ['aseg.mgz', 'aseg.mgz']
>>> fuse.inputs.in_segmentations_noCC = ['aseg.mgz', 'aseg.mgz']
>>> fuse.inputs.in_norms = ['norm.mgz', 'norm.mgz', 'norm.mgz']
>>> fuse.cmdline
'mri_fuse_segmentations -n norm.mgz -a aseg.mgz -c aseg.mgz tp.long.A.template_
↳tp1 tp2'
```

Inputs:

```
[Mandatory]
in_norms: (a list of items which are an existing file name)
    -n <filename> - name of norm file to use (default: norm.mgs) must
    include the corresponding norm file for all given timepoints as well
    as for the current subject
    flag: -n %s
in_segmentations: (a list of items which are an existing file name)
    name of aseg file to use (default: aseg.mgz) must include the aseg
    files for all the given timepoints
    flag: -a %s
in_segmentations_noCC: (a list of items which are an existing file
    name)
    name of aseg file w/o CC labels (default: aseg.auto_noCCseg.mgz)
```

```

        must include the corresponding file for all the given timepoints
        flag: -c %s
    out_file: (a file name)
        output fused segmentation file
    timepoints: (a list of items which are a string)
        subject_ids or timepoints to be processed
        flag: %s, position: -2

[Optional]
    args: (a unicode string)
        Additional parameters to the command
        flag: %s
    environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipype default value: {})
        Environment variables
    ignore_exception: (a boolean, nipype default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
    subject_id: (a string)
        subject_id being processed
        flag: %s, position: -3
    subjects_dir: (an existing directory name)
        subjects directory
    terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
        output fused segmentation file

```

RobustTemplate

[Link to code](#)**Wraps command `mri_robust_template`**

construct an unbiased robust template for longitudinal volumes

Examples

```

>>> from nipype.interfaces.freesurfer import RobustTemplate
>>> template = RobustTemplate()
>>> template.inputs.in_files = ['structural.nii', 'functional.nii']
>>> template.inputs.auto_detect_sensitivity = True
>>> template.inputs.average_metric = 'mean'
>>> template.inputs.initial_timepoint = 1
>>> template.inputs.fixed_timepoint = True
>>> template.inputs.no_iteration = True
>>> template.inputs.subsample_threshold = 200
>>> template.cmdline
'mri_robust_template --satit --average 0 --fixtp --mov structural.nii functional.
↪nii --inittp 1 --noit --template mri_robust_template_out.mgz --subsample 200'
>>> template.inputs.out_file = 'T1.nii'
>>> template.cmdline
'mri_robust_template --satit --average 0 --fixtp --mov structural.nii functional.
↪nii --inittp 1 --noit --template T1.nii --subsample 200'

```

```
>>> template.inputs.transform_outputs = ['structural.lta', 'functional.lta']
>>> template.inputs.scaled_intensity_outputs = ['structural-iscale.txt',
↳ 'functional-iscale.txt']
>>> template.cmdline
'mri_robust_template --satit --average 0 --fixtp --mov structural.nii functional.
↳ nii --inittp 1 --noit --template T1.nii --iscaleout structural-iscale.txt_
↳ functional-iscale.txt --subsample 200 --lta structural.lta functional.lta'
```

```
>>> template.run()
```

References

[https://surfer.nmr.mgh.harvard.edu/fswiki/mri_robust_template]

Inputs:

```
[Mandatory]
auto_detect_sensitivity: (a boolean)
    auto-detect good sensitivity (recommended for head or full brain
    scans)
    flag: --satit
    mutually_exclusive: outlier_sensitivity
in_files: (a list of items which are an existing file name)
    input movable volumes to be aligned to common mean/median template
    flag: --mov %s
out_file: (a file name, nipyre default value:
    mri_robust_template_out.mgz)
    output template volume (final mean/median image)
    flag: --template %s
outlier_sensitivity: (a float)
    set outlier sensitivity manually (e.g. "--sat 4.685" ). Higher
    values mean less sensitivity.
    flag: --sat %.4f
    mutually_exclusive: auto_detect_sensitivity

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
average_metric: ('median' or 'mean')
    construct template from: 0 Mean, 1 Median (default)
    flag: --average %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
fixed_timepoint: (a boolean)
    map everthing to init TP# (init TP is not resampled)
    flag: --fixtp
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_intensity_scales: (a list of items which are an existing file
    name)
    use initial intensity scales
    flag: --iscalein %s
initial_timepoint: (an integer (int or long))
```

```
    use TP# for spacial init (default random), 0: no init
    flag: --inittp %d
initial_transforms: (a list of items which are an existing file name)
    use initial transforms (lta) on source
    flag: --ixforms %s
intensity_scaling: (a boolean)
    allow also intensity scaling (default off)
    flag: --iscale
no_iteration: (a boolean)
    do not iterate, just create first template
    flag: --noit
scaled_intensity_outputs: (a list of items which are a file name)
    final intensity scales (will activate --iscale)
    flag: --iscaleout %s
subjects_dir: (an existing directory name)
    subjects directory
subsample_threshold: (an integer (int or long))
    subsample if dim > # on all axes (default no subs.)
    flag: --subsample %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transform_outputs: (a list of items which are a file name)
    output xforms to template (for each input)
    flag: --lta %s
```

Outputs:

```
out_file: (an existing file name)
    output template volume (final mean/median image)
scaled_intensity_outputs: (a list of items which are an existing file
    name)
    output final intensity scales
transform_outputs: (a list of items which are an existing file name)
    output xform files from moving to template
```

interfaces.freesurfer.model

Binarize

[Link to code](#)

Wraps command **mri_binarize**

Use FreeSurfer mri_binarize to threshold an input volume

Examples

```
>>> binvol = Binarize(in_file='structural.nii', min=10, binary_file='foo_out.nii')
>>> binvol.cmdline
'mri_binarize --o foo_out.nii --i structural.nii --min 10.000000'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input volume
```

```

        flag: --i %s

[Optional]
abs: (a boolean)
    take abs of invol first (ie, make unsigned)
    flag: --abs
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bin_col_num: (a boolean)
    set binarized voxel value to its column number
    flag: --bincol
bin_val: (an integer (int or long))
    set vox within thresh to val (default is 1)
    flag: --binval %d
bin_val_not: (an integer (int or long))
    set vox outside range to val (default is 0)
    flag: --binvalnot %d
binary_file: (a file name)
    binary output volume
    flag: --o %s
count_file: (a boolean or a file name)
    save number of hits in ascii file (hits, ntotvox, pct)
    flag: --count %s
dilate: (an integer (int or long))
    niters: dilate binarization in 3D
    flag: --dilate %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyype default value: {})
    Environment variables
erode: (an integer (int or long))
    nerode: erode binarization in 3D (after any dilation)
    flag: --erode %d
erode2d: (an integer (int or long))
    nerode2d: erode binarization in 2D (after any 3D erosion)
    flag: --erode2d %d
frame_no: (an integer (int or long))
    use 0-based frame of input (default is 0)
    flag: --frame %s
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert: (a boolean)
    set binval=0, binvalnot=1
    flag: --inv
mask_file: (an existing file name)
    must be within mask
    flag: --mask maskvol
mask_thresh: (a float)
    set thresh for mask
    flag: --mask-thresh %f
match: (a list of items which are an integer (int or long))
    match instead of threshold
    flag: --match %d...
max: (a float)
    max thresh
    flag: --max %f

```

```

    mutually_exclusive: wm_ven_csf
merge_file: (an existing file name)
    merge with mergevol
    flag: --merge %s
min: (a float)
    min thresh
    flag: --min %f
    mutually_exclusive: wm_ven_csf
out_type: ('nii' or 'nii.gz' or 'mgz')
    output file type
rmax: (a float)
    compute max based on rmax*globalmean
    flag: --rmax %f
rmin: (a float)
    compute min based on rmin*globalmean
    flag: --rmin %f
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
ventricles: (a boolean)
    set match vals those for aseg ventricles+choroid (not 4th)
    flag: --ventricles
wm: (a boolean)
    set match vals to 2 and 41 (aseg for cerebral WM)
    flag: --wm
wm_ven_csf: (a boolean)
    WM and ventricular CSF, including choroid (not 4th)
    flag: --wm+vcsf
    mutually_exclusive: min, max
zero_edges: (a boolean)
    zero the edge voxels
    flag: --zero-edges
zero_slice_edge: (a boolean)
    zero the edge slice voxels
    flag: --zero-slice-edges

```

Outputs:

```

binary_file: (an existing file name)
    binarized output volume
count_file: (a file name)
    ascii file containing number of hits

```

Concatenate

[Link to code](#)**Wraps command `mri_concat`**

Use Freesurfer `mri_concat` to combine several input volumes into one output volume. Can concatenate by frames, or compute a variety of statistics on the input volumes.

Examples

Combine two input volumes into one volume with two frames

```
>>> concat = Concatenate()
>>> concat.inputs.in_files = ['cont1.nii', 'cont2.nii']
>>> concat.inputs.concatenated_file = 'bar.nii'
>>> concat.cmdline
'mri_concat --o bar.nii --i cont1.nii --i cont2.nii'
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
    Individual volumes to be concatenated
    flag: --i %s...

[Optional]
add_val: (a float)
    Add some amount to the input volume
    flag: --add %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
combine: (a boolean)
    Combine non-zero values into single frame volume
    flag: --combine
concatenated_file: (a file name)
    Output volume
    flag: --o %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
gmean: (an integer (int or long))
    create matrix to average Ng groups, Nper=Ntot/Ng
    flag: --gmean %d
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
keep_dtype: (a boolean)
    Keep voxelwise precision type (default is float
    flag: --keep-datatype
mask_file: (an existing file name)
    Mask input with a volume
    flag: --mask %s
max_bonfcor: (a boolean)
    Compute max and bonferroni correct (assumes -log10(ps))
    flag: --max-bonfcor
max_index: (a boolean)
    Compute the index of max voxel in concatenated volumes
    flag: --max-index
mean_div_n: (a boolean)
    compute mean/nframes (good for var)
    flag: --mean-div-n
multiply_by: (a float)
    Multiply input volume by some amount
    flag: --mul %f
multiply_matrix_file: (an existing file name)
    Multiply input by an ascii matrix in file
    flag: --mtx %s
paired_stats: ('sum' or 'avg' or 'diff' or 'diff-norm' or 'diff-
```

```

    norm1' or 'diff-norm2')
    Compute paired sum, avg, or diff
    flag: --paired-%s
sign: ('abs' or 'pos' or 'neg')
    Take only pos or neg voxles from input, or take abs
    flag: --%s
sort: (a boolean)
    Sort each voxel by ascending frame value
    flag: --sort
stats: ('sum' or 'var' or 'std' or 'max' or 'min' or 'mean')
    Compute the sum, var, std, max, min or mean of the input volumes
    flag: --%s
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
vote: (a boolean)
    Most frequent value at each voxel and fraction of occurances
    flag: --vote

```

Outputs:

```

concatenated_file: (an existing file name)
    Path/name of the output volume

```

GLMFit[Link to code](#)Wraps command **mri_glmfit**Use FreeSurfer's **mri_glmfit** to specify and estimate a general linear model.**Examples**

```

>>> glmfit = GLMFit()
>>> glmfit.inputs.in_file = 'functional.nii'
>>> glmfit.inputs.one_sample = True
>>> glmfit.cmdline == 'mri_glmfit --glmdir %s --y functional.nii --osgm'%os.
↪getcwd()
True

```

Inputs:

```

[Mandatory]
in_file: (a file name)
    input 4D file
    flag: --y %s

[Optional]
allow_ill_cond: (a boolean)
    allow ill-conditioned design matrices
    flag: --illcond
allow_repeated_subjects: (a boolean)
    allow subject names to repeat in the fsgd file (must appear before
    --fsgd
    flag: --allowsbjrep

```



```

args: (a unicode string)
    Additional parameters to the command
    flag: %s
calc_AR1: (a boolean)
    compute and save temporal AR1 of residual
    flag: --tar1
check_opts: (a boolean)
    don't run anything, just check options and exit
    flag: --checkopts
compute_log_y: (a boolean)
    compute natural log of y prior to analysis
    flag: --logy
contrast: (a list of items which are an existing file name)
    contrast file
    flag: --C %s...
cortex: (a boolean)
    use subjects ?h.cortex.label as label
    flag: --cortex
    mutually_exclusive: label_file
debug: (a boolean)
    turn on debugging
    flag: --debug
design: (an existing file name)
    design matrix file
    flag: --X %s
    mutually_exclusive: fsgd, design, one_sample
diag: (an integer (int or long))
    Gdiag_no : set diagnositc level
diag_cluster: (a boolean)
    save sig volume and exit from first sim loop
    flag: --diag-cluster
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixed_fx_dof: (an integer (int or long))
    dof for fixed effects analysis
    flag: --ffxdof %d
    mutually_exclusive: fixed_fx_dof_file
fixed_fx_dof_file: (a file name)
    text file with dof for fixed effects analysis
    flag: --ffxdofdat %d
    mutually_exclusive: fixed_fx_dof
fixed_fx_var: (an existing file name)
    for fixed effects analysis
    flag: --yffxvar %s
force_perm: (a boolean)
    force perumtation test, even when design matrix is not orthog
    flag: --perm-force
fsgd: (a tuple of the form: (an existing file name, 'doss' or
    'dods'))
    freesurfer descriptor file
    flag: --fsgd %s %s
    mutually_exclusive: fsgd, design, one_sample
fwhm: (a floating point number >= 0.0)
    smooth input by fwhm
    flag: --fwhm %f
glm_dir: (a unicode string)

```

```
    save outputs to dir
    flag: --glmdir %s
hemi: ('lh' or 'rh')
    surface hemisphere
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert_mask: (a boolean)
    invert mask
    flag: --mask-inv
label_file: (an existing file name)
    use label as mask, surfaces only
    flag: --label %s
    mutually_exclusive: cortex
mask_file: (an existing file name)
    binary mask
    flag: --mask %s
no_contrast_ok: (a boolean)
    do not fail if no contrasts specified
    flag: --no-contrasts-ok
no_est_fwhm: (a boolean)
    turn off FWHM output estimation
    flag: --no-est-fwhm
no_mask_smooth: (a boolean)
    do not mask when smoothing
    flag: --no-mask-smooth
no_prune: (a boolean)
    do not prune
    flag: --no-prune
    mutually_exclusive: prunethresh
one_sample: (a boolean)
    construct X and C as a one-sample group mean
    flag: --osgm
    mutually_exclusive: one_sample, fsgd, design, contrast
pca: (a boolean)
    perform pca/sgd analysis on residual
    flag: --pca
per_voxel_reg: (a list of items which are an existing file name)
    per-voxel regressors
    flag: --pvr %s...
profile: (an integer (int or long))
    niters : test speed
    flag: --profile %d
prune: (a boolean)
    remove voxels that do not have a non-zero value at each frame (def)
    flag: --prune
prune_thresh: (a float)
    prune threshold. Default is FLT_MIN
    flag: --prune_thr %f
    mutually_exclusive: noprune
resynth_test: (an integer (int or long))
    test GLM by resynthesis
    flag: --resynthtest %d
save_cond: (a boolean)
    flag to save design matrix condition at each voxel
    flag: --save-cond
save_estimate: (a boolean)
    save signal estimate (yhat)
```

```

    flag: --yhat-save
save_res_corr_mtx: (a boolean)
    save residual error spatial correlation matrix (eres.scm). Big!
    flag: --eres-scm
save_residual: (a boolean)
    save residual error (eres)
    flag: --eres-save
seed: (an integer (int or long))
    used for synthesizing noise
    flag: --seed %d
self_reg: (a tuple of the form: (an integer (int or long), an integer
    (int or long), an integer (int or long)))
    self-regressor from index col row slice
    flag: --selfreg %d %d %d
sim_done_file: (a file name)
    create file when simulation finished
    flag: --sim-done %s
sim_sign: ('abs' or 'pos' or 'neg')
    abs, pos, or neg
    flag: --sim-sign %s
simulation: (a tuple of the form: ('perm' or 'mc-full' or 'mc-z', an
    integer (int or long), a float, a unicode string))
    nulltype nsim thresh csdbasename
    flag: --sim %s %d %f %s
subject_id: (a unicode string)
    subject id for surface geometry
subjects_dir: (an existing directory name)
    subjects directory
surf: (a boolean)
    analysis is on a surface mesh
    flag: --surf %s %s %s
    requires: subject_id, hemi
surf_geo: (a unicode string, nipy default value: white)
    surface geometry name (e.g. white, pial)
synth: (a boolean)
    replace input with gaussian
    flag: --synth
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
uniform: (a tuple of the form: (a float, a float))
    use uniform distribution instead of gaussian
    flag: --uniform %f %f
var_fwhm: (a floating point number >= 0.0)
    smooth variance by fwhm
    flag: --var-fwhm %f
vox_dump: (a tuple of the form: (an integer (int or long), an integer
    (int or long), an integer (int or long)))
    dump voxel GLM and exit
    flag: --voxdump %d %d %d
weight_file: (an existing file name)
    weight for each input at each voxel
    mutually_exclusive: weighted_ls
weight_inv: (a boolean)
    invert weights
    flag: --w-inv
    mutually_exclusive: weighted_ls

```

```
weight_sqrt: (a boolean)
    sqrt of weights
    flag: --w-sqrt
    mutually_exclusive: weighted_ls
weighted_ls: (an existing file name)
    weighted least squares
    flag: --wls %s
    mutually_exclusive: weight_file, weight_inv, weight_sqrt
```

Outputs:

```
beta_file: (an existing file name)
    map of regression coefficients
dof_file: (a file name)
    text file with effective degrees-of-freedom for the analysis
error_file: (a file name)
    map of residual error
error_stddev_file: (a file name)
    map of residual error standard deviation
error_var_file: (a file name)
    map of residual error variance
estimate_file: (a file name)
    map of the estimated Y values
frame_eigenvectors: (a file name)
    matrix of frame eigenvectors from residual PCA
ftest_file: (a list of items which are any value)
    map of test statistic values
fwhm_file: (a file name)
    text file with estimated smoothness
gamma_file: (a list of items which are any value)
    map of contrast of regression coefficients
gamma_var_file: (a list of items which are any value)
    map of regression contrast variance
glm_dir: (an existing directory name)
    output directory
mask_file: (a file name)
    map of the mask used in the analysis
sig_file: (a list of items which are any value)
    map of F-test significance (in -log10p)
singular_values: (a file name)
    matrix singular values from residual PCA
spatial_eigenvectors: (a file name)
    map of spatial eigenvectors from residual PCA
svd_stats_file: (a file name)
    text file summarizing the residual PCA
```

Label2Annot

[Link to code](#)

Wraps command **mrisc_label2annot**

Converts a set of surface labels to an annotation file

Examples

```
>>> from nipyype.interfaces.freesurfer import Label2Annot
>>> l2a = Label2Annot()
>>> l2a.inputs.hemisphere = 'lh'
```

```
>>> l2a.inputs.subject_id = '10335'
>>> l2a.inputs.in_labels = ['lh.aparc.label']
>>> l2a.inputs.orig = 'lh.pial'
>>> l2a.inputs.out_annot = 'test'
>>> l2a.cmdline
'mris_label2annot --hemi lh --l lh.aparc.label --a test --s 10335'
```

Inputs:

```
[Mandatory]
hemisphere: ('lh' or 'rh')
    Input hemisphere
    flag: --hemi %s
in_labels: (a list of items which are any value)
    List of input label files
    flag: --l %s...
orig: (an existing file name)
    implicit {hemisphere}.orig
out_annot: (a string)
    Name of the annotation to create
    flag: --a %s
subject_id: (a string, nipype default value: subject_id)
    Subject name/ID
    flag: --s %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
color_table: (an existing file name)
    File that defines the structure names, their indices, and their
    color
    flag: --ctab %s
copy_inputs: (a boolean)
    copy implicit inputs and create a temp subjects_dir
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
keep_max: (a boolean)
    Keep label with highest 'stat' value
    flag: --maxstatwinner
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose_off: (a boolean)
    Turn off overlap and stat override messages
    flag: --noverbose
```

Outputs:

```
out_file: (an existing file name)
    Output annotation file
```

Label2Label

[Link to code](#)

Wraps command **mri_label2label**

Converts a label in one subject's space to a label in another subject's space using either talairach or spherical as an intermediate registration space.

If a source mask is used, then the input label must have been created from a surface (ie, the vertex numbers are valid). The format can be anything supported by mri_convert or curv or paint. Vertices in the source label that do not meet threshold in the mask will be removed from the label.

Examples

```

>>> from nipy.interfaces.freesurfer import Label2Label
>>> l2l = Label2Label()
>>> l2l.inputs.hemisphere = 'lh'
>>> l2l.inputs.subject_id = '10335'
>>> l2l.inputs.sphere_reg = 'lh.pial'
>>> l2l.inputs.white = 'lh.pial'
>>> l2l.inputs.source_subject = 'fsaverage'
>>> l2l.inputs.source_label = 'lh-pial.stl'
>>> l2l.inputs.source_white = 'lh.pial'
>>> l2l.inputs.source_sphere_reg = 'lh.pial'
>>> l2l.cmdline
'mri_label2label --hemi lh --trglabel lh-pial_converted.stl --regmethod surface --
↳srclabel lh-pial.stl --srcsubject fsaverage --trgsubject 10335'

```

Inputs:

```

[Mandatory]
hemisphere: ('lh' or 'rh')
    Input hemisphere
    flag: --hemi %s
source_label: (an existing file name)
    Source label
    flag: --srclabel %s
source_sphere_reg: (an existing file name)
    Implicit input <hemisphere>.sphere.reg
source_subject: (a string)
    Source subject name
    flag: --srcsubject %s
source_white: (an existing file name)
    Implicit input <hemisphere>.white
sphere_reg: (an existing file name)
    Implicit input <hemisphere>.sphere.reg
subject_id: (a string, nipy default value: subject_id)
    Target subject
    flag: --trgsubject %s
white: (an existing file name)
    Implicit input <hemisphere>.white

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the input
    files to the node directory.
environ: (a dictionary with keys which are a bytes or None or a value

```

```

        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    Target label
    flag: --trglabel %s
registration_method: ('surface' or 'volume', nipy default value:
    surface)
    Registration method
    flag: --regmethod %s
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    Output label

```

Label2Vol[Link to code](#)Wraps command **mri_label2vol**

Make a binary volume from a Freesurfer label

Examples

```

>>> binvol = Label2Vol(label_file='cortex.label', template_file='structural.nii',
↳reg_file='register.dat', fill_thresh=0.5, vol_label_file='foo_out.nii')
>>> binvol.cmdline
'mri_label2vol --fillthresh 0 --label cortex.label --reg register.dat --temp
↳structural.nii --o foo_out.nii'

```

Inputs:

```

[Mandatory]
annot_file: (an existing file name)
    surface annotation file
    flag: --annot %s
    mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg
    requires: subject_id, hemi
aparc_aseg: (a boolean)
    use aparc+aseg.mgz in subjectdir as seg
    flag: --aparc+aseg
    mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg
label_file: (a list of items which are an existing file name)
    list of label files
    flag: --label %s...
    mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg
seg_file: (an existing file name)
    segmentation file

```

```
    flag: --seg %s
    mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg
template_file: (an existing file name)
    output template volume
    flag: --temp %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyype default value: {})
    Environment variables
fill_thresh: (0.0 <= a floating point number <= 1.0)
    thresh : between 0 and 1
    flag: --fillthresh %.f
hemi: ('lh' or 'rh')
    hemisphere to use lh or rh
    flag: --hemi %s
identity: (a boolean)
    set R=I
    flag: --identity
    mutually_exclusive: reg_file, reg_header, identity
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert_mtx: (a boolean)
    Invert the registration matrix
    flag: --invertmtx
label_hit_file: (a file name)
    file with each frame is nhits for a label
    flag: --hits %s
label_voxel_volume: (a float)
    volume of each label point (def 1mm3)
    flag: --labvoxvol %f
map_label_stat: (a file name)
    map the label stats field into the vol
    flag: --label-stat %s
native_vox2ras: (a boolean)
    use native vox2ras xform instead of tkregister-style
    flag: --native-vox2ras
proj: (a tuple of the form: ('abs' or 'frac', a float, a float, a
    float))
    project along surface normal
    flag: --proj %s %f %f %f
    requires: subject_id, hemi
reg_file: (an existing file name)
    tkregister style matrix VolXYZ = R*LabelXYZ
    flag: --reg %s
    mutually_exclusive: reg_file, reg_header, identity
reg_header: (an existing file name)
    label template volume
    flag: --regheader %s
    mutually_exclusive: reg_file, reg_header, identity
subject_id: (a unicode string)
    subject id
    flag: --subject %s
```



```

subjects_dir: (an existing directory name)
               subjects directory
surface: (a unicode string)
          use surface instead of white
          flag: --surf %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
vol_label_file: (a file name)
                 output volume
                 flag: --o %s

```

Outputs:

```

vol_label_file: (an existing file name)
                 output volume

```

MRISPreproc[Link to code](#)**Wraps command `mriss_preproc`**Use FreeSurfer `mriss_preproc` to prepare a group of contrasts for a second level analysis**Examples**

```

>>> preproc = MRISPreproc()
>>> preproc.inputs.target = 'fsaverage'
>>> preproc.inputs.hemi = 'lh'
>>> preproc.inputs.vol_measure_file = [('cont1.nii', 'register.dat'),
    ↪                                ('cont1a.nii', 'register.dat')]
>>> preproc.inputs.out_file = 'concatenated_file.mgz'
>>> preproc.cmdline
'mriss_preproc --hemi lh --out concatenated_file.mgz --target fsaverage --iv cont1.
    ↪nii register.dat --iv cont1a.nii register.dat'

```

Inputs:

```

[Mandatory]
hemi: ('lh' or 'rh')
      hemisphere for source and target
      flag: --hemi %s
target: (a unicode string)
        target subject name
        flag: --target %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
fsgd_file: (an existing file name)
            specify subjects using fsgd file
            flag: --fsgd %s

```

```

    mutually_exclusive: subjects, fsgd_file, subject_file
fwhm: (a float)
    smooth by fwhm mm on the target surface
    flag: --fwhm %f
    mutually_exclusive: num_iters
fwhm_source: (a float)
    smooth by fwhm mm on the source surface
    flag: --fwhm-src %f
    mutually_exclusive: num_iters_source
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
num_iters: (an integer (int or long))
    niters : smooth by niters on the target surface
    flag: --niters %d
    mutually_exclusive: fwhm
num_iters_source: (an integer (int or long))
    niters : smooth by niters on the source surface
    flag: --niterssrc %d
    mutually_exclusive: fwhm_source
out_file: (a file name)
    output filename
    flag: --out %s
proj_frac: (a float)
    projection fraction for vol2surf
    flag: --projfrac %s
smooth_cortex_only: (a boolean)
    only smooth cortex (ie, exclude medial wall)
    flag: --smooth-cortex-only
source_format: (a unicode string)
    source format
    flag: --srcfmt %s
subject_file: (an existing file name)
    file specifying subjects separated by white space
    flag: --f %s
    mutually_exclusive: subjects, fsgd_file, subject_file
subjects: (a list of items which are any value)
    subjects from who measures are calculated
    flag: --s %s...
    mutually_exclusive: subjects, fsgd_file, subject_file
subjects_dir: (an existing directory name)
    subjects directory
surf_area: (a unicode string)
    Extract vertex area from subject/surf/hemi.surfname to use as input.
    flag: --area %s
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
surf_dir: (a unicode string)
    alternative directory (instead of surf)
    flag: --surfdir %s
surf_measure: (a unicode string)
    Use subject/surf/hemi.surf_measure as input
    flag: --meas %s
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
surf_measure_file: (a list of items which are an existing file name)
    file alternative to surfmeas, still requires list of subjects
    flag: --is %s...
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')

```

```
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
vol_measure_file: (a list of items which are a tuple of the form: (an
    existing file name, an existing file name))
list of volume measure and reg file tuples
flag: --iv %s %s...
```

Outputs:

```
out_file: (a file name)
    preprocessed output file
```

MRISPreprocReconAll[Link to code](#)Wraps command **mriss_preproc**

Extends MRISPreproc to allow it to be used in a recon-all workflow

Examples

```
>>> preproc = MRISPreprocReconAll()
>>> preproc.inputs.target = 'fsaverage'
>>> preproc.inputs.hemi = 'lh'
>>> preproc.inputs.vol_measure_file = [('cont1.nii', 'register.dat'),
    ↪ ('cont1a.nii', 'register.dat')]
>>> preproc.inputs.out_file = 'concatenated_file.mgz'
>>> preproc.cmdline
'mris_preproc --hemi lh --out concatenated_file.mgz --s subject_id --target_
↪ fsaverage --iv cont1.nii register.dat --iv cont1a.nii register.dat'
```

Inputs:

```
[Mandatory]
hemi: ('lh' or 'rh')
    hemisphere for source and target
    flag: --hemi %s
target: (a unicode string)
    target subject name
    flag: --target %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
copy_inputs: (a boolean)
    If running as a node, set this to True this will copy some implicit
    inputs to the node directory.
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fsgd_file: (an existing file name)
    specify subjects using fsgd file
    flag: --fsgd %s
    mutually_exclusive: subjects, fsgd_file, subject_file
fwhm: (a float)
```

```
    smooth by fwhm mm on the target surface
    flag: --fwhm %f
    mutually_exclusive: num_iters
fwhm_source: (a float)
    smooth by fwhm mm on the source surface
    flag: --fwhm-src %f
    mutually_exclusive: num_iters_source
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
lh_surfreg_target: (a file name)
    Implicit target surface registration file
    requires: surfreg_files
num_iters: (an integer (int or long))
    niters : smooth by niters on the target surface
    flag: --niters %d
    mutually_exclusive: fwhm
num_iters_source: (an integer (int or long))
    niters : smooth by niters on the source surface
    flag: --niterssrc %d
    mutually_exclusive: fwhm_source
out_file: (a file name)
    output filename
    flag: --out %s
proj_frac: (a float)
    projection fraction for vol2surf
    flag: --projfrac %s
rh_surfreg_target: (a file name)
    Implicit target surface registration file
    requires: surfreg_files
smooth_cortex_only: (a boolean)
    only smooth cortex (ie, exclude medial wall)
    flag: --smooth-cortex-only
source_format: (a unicode string)
    source format
    flag: --srcfmt %s
subject_file: (an existing file name)
    file specifying subjects separated by white space
    flag: --f %s
    mutually_exclusive: subjects, fsgd_file, subject_file
subject_id: (a string, nipy default value: subject_id)
    subject from whom measures are calculated
    flag: --s %s
    mutually_exclusive: subjects, fsgd_file, subject_file, subject_id
subjects: (a list of items which are any value)
    subjects from who measures are calculated
    flag: --s %s...
    mutually_exclusive: subjects, fsgd_file, subject_file
subjects_dir: (an existing directory name)
    subjects directory
surf_area: (a unicode string)
    Extract vertex area from subject/surf/hemi.surfname to use as input.
    flag: --area %s
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
surf_dir: (a unicode string)
    alternative directory (instead of surf)
    flag: --surfdir %s
surf_measure: (a unicode string)
```

```

    Use subject/surf/hemi.surf_measure as input
    flag: --meas %s
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
surf_measure_file: (an existing file name)
    file necessary for surfmeas
    flag: --meas %s
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
surfreg_files: (a list of items which are an existing file name)
    lh and rh input surface registration files
    flag: --surfreg %s
    requires: lh_surfreg_target, rh_surfreg_target
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
vol_measure_file: (a list of items which are a tuple of the form: (an
    existing file name, an existing file name))
    list of volume measure and reg file tuples
    flag: --iv %s %s...

```

Outputs:

```

out_file: (a file name)
    preprocessed output file

```

MS_LDA[Link to code](#)Wraps command **mri_ms_LDA**

Perform LDA reduction on the intensity space of an arbitrary # of FLASH images

Examples

```

>>> grey_label = 2
>>> white_label = 3
>>> zero_value = 1
>>> optimalWeights = MS_LDA(lda_labels=[grey_label, white_label],
↳          label_file='label.mgz', weight_file='weights.txt',
↳          shift=zero_value, vol_synth_file='synth_out.mgz',
↳          conform=True, use_weights=True,
↳          images=['FLASH1.mgz', 'FLASH2.mgz', 'FLASH3.mgz'])
>>> optimalWeights.cmdline
'mri_ms_LDA -conform -label label.mgz -lda 2 3 -shift 1 -W -synth synth_out.mgz -
↳ weight weights.txt FLASH1.mgz FLASH2.mgz FLASH3.mgz'

```

Inputs:

```

[Mandatory]
images: (a list of items which are an existing file name)
    list of input FLASH images
    flag: %s, position: -1
lda_labels: (a list of from 2 to 2 items which are an integer (int or
    long))
    pair of class labels to optimize
    flag: -lda %s
vol_synth_file: (a file name)
    filename for the synthesized output volume

```

```
    flag: -synth %s
weight_file: (a file name)
    filename for the LDA weights (input or output)
    flag: -weight %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
conform: (a boolean)
    Conform the input volumes (brain mask typically already conformed)
    flag: -conform
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
label_file: (a file name)
    filename of the label volume
    flag: -label %s
mask_file: (a file name)
    filename of the brain mask volume
    flag: -mask %s
shift: (an integer (int or long))
    shift all values equal to the given value to zero
    flag: -shift %d
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
use_weights: (a boolean)
    Use the weights from a previously generated weight file
    flag: -W
```

Outputs:

```
vol_synth_file: (an existing file name)
weight_file: (an existing file name)
```

OneSampleTTest

[Link to code](#)

Wraps command **mri_glmfit**

Inputs:

```
[Mandatory]
in_file: (a file name)
    input 4D file
    flag: --y %s

[Optional]
allow_ill_cond: (a boolean)
    allow ill-conditioned design matrices
    flag: --illcond
```

```

allow_repeated_subjects: (a boolean)
    allow subject names to repeat in the fsgd file (must appear before
    --fsgd
    flag: --allowsubprep
args: (a unicode string)
    Additional parameters to the command
    flag: %s
calc_AR1: (a boolean)
    compute and save temporal AR1 of residual
    flag: --tar1
check_opts: (a boolean)
    don't run anything, just check options and exit
    flag: --checkopts
compute_log_y: (a boolean)
    compute natural log of y prior to analysis
    flag: --logy
contrast: (a list of items which are an existing file name)
    contrast file
    flag: --C %s...
cortex: (a boolean)
    use subjects ?h.cortex.label as label
    flag: --cortex
    mutually_exclusive: label_file
debug: (a boolean)
    turn on debugging
    flag: --debug
design: (an existing file name)
    design matrix file
    flag: --X %s
    mutually_exclusive: fsgd, design, one_sample
diag: (an integer (int or long))
    Gdiag_no : set diagnositc level
diag_cluster: (a boolean)
    save sig volume and exit from first sim loop
    flag: --diag-cluster
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
fixed_fx_dof: (an integer (int or long))
    dof for fixed effects analysis
    flag: --ffxdof %d
    mutually_exclusive: fixed_fx_dof_file
fixed_fx_dof_file: (a file name)
    text file with dof for fixed effects analysis
    flag: --ffxdofdat %d
    mutually_exclusive: fixed_fx_dof
fixed_fx_var: (an existing file name)
    for fixed effects analysis
    flag: --yffxvar %s
force_perm: (a boolean)
    force perumtation test, even when design matrix is not orthog
    flag: --perm-force
fsgd: (a tuple of the form: (an existing file name, 'doss' or
    'dods'))
    freesurfer descriptor file
    flag: --fsgd %s %s
    mutually_exclusive: fsgd, design, one_sample

```

```
fwhm: (a floating point number >= 0.0)
    smooth input by fwhm
    flag: --fwhm %f
glm_dir: (a unicode string)
    save outputs to dir
    flag: --glmdir %s
hemi: ('lh' or 'rh')
    surface hemisphere
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert_mask: (a boolean)
    invert mask
    flag: --mask-inv
label_file: (an existing file name)
    use label as mask, surfaces only
    flag: --label %s
    mutually_exclusive: cortex
mask_file: (an existing file name)
    binary mask
    flag: --mask %s
no_contrast_ok: (a boolean)
    do not fail if no contrasts specified
    flag: --no-contrasts-ok
no_est_fwhm: (a boolean)
    turn off FWHM output estimation
    flag: --no-est-fwhm
no_mask_smooth: (a boolean)
    do not mask when smoothing
    flag: --no-mask-smooth
no_prune: (a boolean)
    do not prune
    flag: --no-prune
    mutually_exclusive: prunethresh
one_sample: (a boolean)
    construct X and C as a one-sample group mean
    flag: --osgm
    mutually_exclusive: one_sample, fsgd, design, contrast
pca: (a boolean)
    perform pca/svd analysis on residual
    flag: --pca
per_voxel_reg: (a list of items which are an existing file name)
    per-voxel regressors
    flag: --pvr %s...
profile: (an integer (int or long))
    niters : test speed
    flag: --profile %d
prune: (a boolean)
    remove voxels that do not have a non-zero value at each frame (def)
    flag: --prune
prune_thresh: (a float)
    prune threshold. Default is FLT_MIN
    flag: --prune_thr %f
    mutually_exclusive: noprun
resynth_test: (an integer (int or long))
    test GLM by resynthesis
    flag: --resynthtest %d
save_cond: (a boolean)
```



```

        flag to save design matrix condition at each voxel
    flag: --save-cond
    save_estimate: (a boolean)
        save signal estimate (yhat)
    flag: --yhat-save
    save_res_corr_mtx: (a boolean)
        save residual error spatial correlation matrix (eres.scm). Big!
    flag: --eres-scm
    save_residual: (a boolean)
        save residual error (eres)
    flag: --eres-save
    seed: (an integer (int or long))
        used for synthesizing noise
    flag: --seed %d
    self_reg: (a tuple of the form: (an integer (int or long), an integer
        (int or long), an integer (int or long)))
        self-regressor from index col row slice
    flag: --selfreg %d %d %d
    sim_done_file: (a file name)
        create file when simulation finished
    flag: --sim-done %s
    sim_sign: ('abs' or 'pos' or 'neg')
        abs, pos, or neg
    flag: --sim-sign %s
    simulation: (a tuple of the form: ('perm' or 'mc-full' or 'mc-z', an
        integer (int or long), a float, a unicode string))
        nulltype nsim thresh csdbasename
    flag: --sim %s %d %f %s
    subject_id: (a unicode string)
        subject id for surface geometry
    subjects_dir: (an existing directory name)
        subjects directory
    surf: (a boolean)
        analysis is on a surface mesh
    flag: --surf %s %s %s
        requires: subject_id, hemi
    surf_geo: (a unicode string, nipy default value: white)
        surface geometry name (e.g. white, pial)
    synth: (a boolean)
        replace input with gaussian
    flag: --synth
    terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
    uniform: (a tuple of the form: (a float, a float))
        use uniform distribution instead of gaussian
    flag: --uniform %f %f
    var_fwhm: (a floating point number >= 0.0)
        smooth variance by fwhm
    flag: --var-fwhm %f
    vox_dump: (a tuple of the form: (an integer (int or long), an integer
        (int or long), an integer (int or long)))
        dump voxel GLM and exit
    flag: --voxdump %d %d %d
    weight_file: (an existing file name)
        weight for each input at each voxel
        mutually_exclusive: weighted_ls

```

```
weight_inv: (a boolean)
    invert weights
    flag: --w-inv
    mutually_exclusive: weighted_ls
weight_sqrt: (a boolean)
    sqrt of weights
    flag: --w-sqrt
    mutually_exclusive: weighted_ls
weighted_ls: (an existing file name)
    weighted least squares
    flag: --wls %s
    mutually_exclusive: weight_file, weight_inv, weight_sqrt
```

Outputs:

```
beta_file: (an existing file name)
    map of regression coefficients
dof_file: (a file name)
    text file with effective degrees-of-freedom for the analysis
error_file: (a file name)
    map of residual error
error_stddev_file: (a file name)
    map of residual error standard deviation
error_var_file: (a file name)
    map of residual error variance
estimate_file: (a file name)
    map of the estimated Y values
frame_eigenvectors: (a file name)
    matrix of frame eigenvectors from residual PCA
ftest_file: (a list of items which are any value)
    map of test statistic values
fwhm_file: (a file name)
    text file with estimated smoothness
gamma_file: (a list of items which are any value)
    map of contrast of regression coefficients
gamma_var_file: (a list of items which are any value)
    map of regression contrast variance
glm_dir: (an existing directory name)
    output directory
mask_file: (a file name)
    map of the mask used in the analysis
sig_file: (a list of items which are any value)
    map of F-test significance (in -log10p)
singular_values: (a file name)
    matrix singular values from residual PCA
spatial_eigenvectors: (a file name)
    map of spatial eigenvectors from residual PCA
svd_stats_file: (a file name)
    text file summarizing the residual PCA
```

SegStats

[Link to code](#)

Wraps command **mri_segstats**

Use FreeSurfer mri_segstats for ROI analysis

Examples

```
>>> import nipy.interfaces.freesurfer as fs
>>> ss = fs.SegStats()
>>> ss.inputs.annot = ('PWS04', 'lh', 'aparc')
>>> ss.inputs.in_file = 'functional.nii'
>>> ss.inputs.subjects_dir = '.'
>>> ss.inputs.avgwf_txt_file = 'avgwf.txt'
>>> ss.inputs.summary_file = 'summary.stats'
>>> ss.cmdline
'mri_segstats --annot PWS04 lh aparc --avgwf ./avgwf.txt --i functional.nii --sum_
↪ ./summary.stats'
```

Inputs:

```
[Mandatory]
annot: (a tuple of the form: (a unicode string, 'lh' or 'rh', a
    unicode string))
    subject hemi parc : use surface parcellation
    flag: --annot %s %s %s
    mutually_exclusive: segmentation_file, annot, surf_label
segmentation_file: (an existing file name)
    segmentation volume path
    flag: --seg %s
    mutually_exclusive: segmentation_file, annot, surf_label
surf_label: (a tuple of the form: (a unicode string, 'lh' or 'rh', a
    unicode string))
    subject hemi label : use surface label
    flag: --slabel %s %s %s
    mutually_exclusive: segmentation_file, annot, surf_label

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
avgwf_file: (a boolean or a file name)
    Save as binary volume (bool or filename)
    flag: --avgwfvol %s
avgwf_txt_file: (a boolean or a file name)
    Save average waveform into file (bool or filename)
    flag: --avgwf %s
brain_vol: ('brain-vol-from-seg' or 'brainmask')
    Compute brain volume either with ``brainmask`` or ``brain-vol-from-
    seg``
    flag: --%s
brainmask_file: (an existing file name)
    Load brain mask and compute the volume of the brain as the non-zero
    voxels in this volume
    flag: --brainmask %s
calc_power: ('sqr' or 'sqrt')
    Compute either the sqr or the sqrt of the input
    flag: --%s
calc_snr: (a boolean)
    save mean/std as extra column in output table
    flag: --snr
color_table_file: (an existing file name)
    color table file with seg id names
    flag: --ctab %s
    mutually_exclusive: color_table_file, default_color_table,
```

```
    gca_color_table
cortex_vol_from_surf: (a boolean)
    Compute cortex volume from surf
    flag: --surf-ctx-vol
default_color_table: (a boolean)
    use $FREESURFER_HOME/FreeSurferColorLUT.txt
    flag: --ctab-default
    mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
empty: (a boolean)
    Report on segmentations listed in the color table
    flag: --empty
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
etiv: (a boolean)
    Compute ICV from talairach transform
    flag: --etiv
etiv_only: ('etiv' or 'old-etiv' or '--%s-only')
    Compute etiv and exit. Use ``etiv`` or ``old-etiv``
euler: (a boolean)
    Write out number of defect holes in orig.nofix based on the euler
    number
    flag: --euler
exclude_ctx_gm_wm: (a boolean)
    exclude cortical gray and white matter
    flag: --excl-ctxgmwm
exclude_id: (an integer (int or long))
    Exclude seg id from report
    flag: --excludeid %d
frame: (an integer (int or long))
    Report stats on nth frame of input volume
    flag: --frame %d
gca_color_table: (an existing file name)
    get color table from GCA (CMA)
    flag: --ctab-gca %s
    mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    Use the segmentation to report stats on this volume
    flag: --i %s
in_intensity: (a file name)
    Undocumented input norm.mgz file
    flag: --in %s --in-intensity-name %s
intensity_units: ('MR')
    Intensity units
    flag: --in-intensity-units %s
    requires: in_intensity
mask_erode: (an integer (int or long))
    Erode mask by some amount
    flag: --maskerode %d
mask_file: (an existing file name)
    Mask volume (same size as seg
    flag: --mask %s
```

```

mask_frame: (an integer (int or long))
    Mask with this (0 based) frame of the mask volume
    requires: mask_file
mask_invert: (a boolean)
    Invert binarized mask volume
    flag: --maskinvert
mask_sign: ('abs' or 'pos' or 'neg' or '--masksign %s')
    Sign for mask threshold: pos, neg, or abs
mask_thresh: (a float)
    binarize mask with this threshold <0.5>
    flag: --maskthresh %f
multiply: (a float)
    multiply input by val
    flag: --mul %f
non_empty_only: (a boolean)
    Only report nonempty segmentations
    flag: --nonempty
partial_volume_file: (an existing file name)
    Compensate for partial voluming
    flag: --pv %s
segment_id: (a list of items which are any value)
    Manually specify segmentation ids
    flag: --id %s...
sf_avg_file: (a boolean or a file name)
    Save mean across space and time
    flag: --sfavg %s
subcort_gm: (a boolean)
    Compute volume of subcortical gray matter
    flag: --subcortgray
subjects_dir: (an existing directory name)
    subjects directory
summary_file: (a file name)
    Segmentation stats summary table file
    flag: --sum %s, position: -1
supratent: (a boolean)
    Undocumented input flag
    flag: --supratent
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
total_gray: (a boolean)
    Compute volume of total gray matter
    flag: --totalgray
vox: (a list of items which are an integer (int or long))
    Replace seg with all 0s except at C R S (three int inputs)
    flag: --vox %s
wm_vol_from_surf: (a boolean)
    Compute wm volume from surf
    flag: --surf-wm-vol

```

Outputs:

```

avgwf_file: (a file name)
    Volume with functional statistics averaged over segs
avgwf_txt_file: (a file name)
    Text file with functional statistics averaged over segs
sf_avg_file: (a file name)

```

```

Text file with func statistics averaged over segs and framss
summary_file: (an existing file name)
Segmentation summary statistics table

```

SegStatsReconAll

[Link to code](#)

Wraps command `mri_segstats`

This class inherits `SegStats` and modifies it for use in a recon-all workflow. This implementation mandates implicit inputs that `SegStats`. To ensure backwards compatability of `SegStats`, this class was created.

Examples

```

>>> from nipy.interfaces.freesurfer import SegStatsReconAll
>>> segstatsreconall = SegStatsReconAll()
>>> segstatsreconall.inputs.annot = ('PWS04', 'lh', 'aparc')
>>> segstatsreconall.inputs.avgwf_txt_file = 'avgwf.txt'
>>> segstatsreconall.inputs.summary_file = 'summary.stats'
>>> segstatsreconall.inputs.subject_id = '10335'
>>> segstatsreconall.inputs.ribbon = 'wm.mgz'
>>> segstatsreconall.inputs.transform = 'trans.mat'
>>> segstatsreconall.inputs.presurf_seg = 'wm.mgz'
>>> segstatsreconall.inputs.lh_orig_nofix = 'lh.pial'
>>> segstatsreconall.inputs.rh_orig_nofix = 'lh.pial'
>>> segstatsreconall.inputs.lh_pial = 'lh.pial'
>>> segstatsreconall.inputs.rh_pial = 'lh.pial'
>>> segstatsreconall.inputs.lh_white = 'lh.pial'
>>> segstatsreconall.inputs.rh_white = 'lh.pial'
>>> segstatsreconall.inputs.empty = True
>>> segstatsreconall.inputs.brain_vol = 'brain-vol-from-seg'
>>> segstatsreconall.inputs.exclude_ctx_gm_wm = True
>>> segstatsreconall.inputs.supratent = True
>>> segstatsreconall.inputs.subcort_gm = True
>>> segstatsreconall.inputs.etiv = True
>>> segstatsreconall.inputs.wm_vol_from_surf = True
>>> segstatsreconall.inputs.cortex_vol_from_surf = True
>>> segstatsreconall.inputs.total_gray = True
>>> segstatsreconall.inputs.euler = True
>>> segstatsreconall.inputs.exclude_id = 0
>>> segstatsreconall.cmdline
'mri_segstats --annot PWS04 lh aparc --avgwf ./avgwf.txt --brain-vol-from-seg --
↪surf-ctx-vol --empty --etiv --euler --excl-ctxgmwm --excludeid 0 --subcortgray -
↪subject 10335 --supratent --totalgray --surf-wm-vol --sum ./summary.stats'

```

Inputs:

```

[Mandatory]
annot: (a tuple of the form: (a unicode string, 'lh' or 'rh', a
    unicode string))
    subject hemi parc : use surface parcellation
    flag: --annot %s %s %s
    mutually_exclusive: segmentation_file, annot, surf_label
lh_orig_nofix: (an existing file name)
    Input lh.orig.nofix
lh_pial: (an existing file name)
    Input file must be <subject_id>/surf/lh.pial
lh_white: (an existing file name)

```

```

    Input file must be <subject_id>/surf/lh.white
rh_orig_nofix: (an existing file name)
    Input rh.orig.nofix
rh_pial: (an existing file name)
    Input file must be <subject_id>/surf/rh.pial
rh_white: (an existing file name)
    Input file must be <subject_id>/surf/rh.white
ribbon: (a file name)
    Input file mri/ribbon.mgz
segmentation_file: (an existing file name)
    segmentation volume path
    flag: --seg %s
    mutually_exclusive: segmentation_file, annot, surf_label
subject_id: (a string, nipype default value: subject_id)
    Subject id being processed
    flag: --subject %s
surf_label: (a tuple of the form: (a unicode string, 'lh' or 'rh', a
    unicode string))
    subject hemi label : use surface label
    flag: --slabel %s %s %s
    mutually_exclusive: segmentation_file, annot, surf_label
transform: (an existing file name)
    Input transform file

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
aseg: (an existing file name)
    Mandatory implicit input in 5.3
avgwf_file: (a boolean or a file name)
    Save as binary volume (bool or filename)
    flag: --avgwfvol %s
avgwf_txt_file: (a boolean or a file name)
    Save average waveform into file (bool or filename)
    flag: --avgwf %s
brain_vol: ('brain-vol-from-seg' or 'brainmask')
    Compute brain volume either with ``brainmask`` or ``brain-vol-from-
    seg``
    flag: --%s
brainmask_file: (an existing file name)
    Load brain mask and compute the volume of the brain as the non-zero
    voxels in this volume
    flag: --brainmask %s
calc_power: ('sqr' or 'sqrt')
    Compute either the sqr or the sqrt of the input
    flag: --%s
calc_snr: (a boolean)
    save mean/std as extra column in output table
    flag: --snr
color_table_file: (an existing file name)
    color table file with seg id names
    flag: --ctab %s
    mutually_exclusive: color_table_file, default_color_table,
    gca_color_table
copy_inputs: (a boolean)
    If running as a node, set this to True otherwise, this will copy the
    implicit inputs to the node directory.

```

```
cortex_vol_from_surf: (a boolean)
    Compute cortex volume from surf
    flag: --surf-ctx-vol
default_color_table: (a boolean)
    use $FREESURFER_HOME/FreeSurferColorLUT.txt
    flag: --ctab-default
    mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
empty: (a boolean)
    Report on segmentations listed in the color table
    flag: --empty
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
etiv: (a boolean)
    Compute ICV from talairach transform
    flag: --etiv
etiv_only: ('etiv' or 'old-etiv' or '--s-only')
    Compute etiv and exit. Use ``etiv`` or ``old-etiv``
euler: (a boolean)
    Write out number of defect holes in orig.nofix based on the euler
    number
    flag: --euler
exclude_ctx_gm_wm: (a boolean)
    exclude cortical gray and white matter
    flag: --excl-ctxgmwm
exclude_id: (an integer (int or long))
    Exclude seg id from report
    flag: --excludeid %d
frame: (an integer (int or long))
    Report stats on nth frame of input volume
    flag: --frame %d
gca_color_table: (an existing file name)
    get color table from GCA (CMA)
    flag: --ctab-gca %s
    mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    Use the segmentation to report stats on this volume
    flag: --i %s
in_intensity: (a file name)
    Undocumented input norm.mgz file
    flag: --in %s --in-intensity-name %s
intensity_units: ('MR')
    Intensity units
    flag: --in-intensity-units %s
    requires: in_intensity
mask_erode: (an integer (int or long))
    Erode mask by some amount
    flag: --maskerode %d
mask_file: (an existing file name)
    Mask volume (same size as seg
    flag: --mask %s
mask_frame: (an integer (int or long))
```



```

    Mask with this (0 based) frame of the mask volume
    requires: mask_file
mask_invert: (a boolean)
    Invert binarized mask volume
    flag: --maskinvert
mask_sign: ('abs' or 'pos' or 'neg' or '--masksign %s')
    Sign for mask threshold: pos, neg, or abs
mask_thresh: (a float)
    binarize mask with this threshold <0.5>
    flag: --maskthresh %f
multiply: (a float)
    multiply input by val
    flag: --mul %f
non_empty_only: (a boolean)
    Only report nonempty segmentations
    flag: --nonempty
partial_volume_file: (an existing file name)
    Compensate for partial voluming
    flag: --pv %s
presurf_seg: (an existing file name)
    Input segmentation volume
segment_id: (a list of items which are any value)
    Manually specify segmentation ids
    flag: --id %s...
sf_avg_file: (a boolean or a file name)
    Save mean across space and time
    flag: --sfavg %s
subcort_gm: (a boolean)
    Compute volume of subcortical gray matter
    flag: --subcortgray
subjects_dir: (an existing directory name)
    subjects directory
summary_file: (a file name)
    Segmentation stats summary table file
    flag: --sum %s, position: -1
supratent: (a boolean)
    Undocumented input flag
    flag: --supratent
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
total_gray: (a boolean)
    Compute volume of total gray matter
    flag: --totalgray
vox: (a list of items which are an integer (int or long))
    Replace seg with all 0s except at C R S (three int inputs)
    flag: --vox %s
wm_vol_from_surf: (a boolean)
    Compute wm volume from surf
    flag: --surf-wm-vol

```

Outputs:

```

avgwf_file: (a file name)
    Volume with functional statistics averaged over segs
avgwf_txt_file: (a file name)
    Text file with functional statistics averaged over segs

```

```
sf_avg_file: (a file name)
    Text file with func statistics averaged over segs and framss
summary_file: (an existing file name)
    Segmentation summary statistics table
```

SphericalAverage

[Link to code](#)

Wraps command **mrisspherical_average**

This program will add a template into an average surface.

Examples

```
>>> from nipy.interfaces.freesurfer import SphericalAverage
>>> sphericalavg = SphericalAverage()
>>> sphericalavg.inputs.out_file = 'test.out'
>>> sphericalavg.inputs.in_average = '.'
>>> sphericalavg.inputs.in_surf = 'lh.pial'
>>> sphericalavg.inputs.hemisphere = 'lh'
>>> sphericalavg.inputs.fname = 'lh.entorhinal'
>>> sphericalavg.inputs.which = 'label'
>>> sphericalavg.inputs.subject_id = '10335'
>>> sphericalavg.inputs.erode = 2
>>> sphericalavg.inputs.threshold = 5
>>> sphericalavg.cmdline
'mris_spherical_average -erode 2 -o 10335 -t 5.0 label lh.entorhinal lh pial .
↳test.out'
```

Inputs:

```
[Mandatory]
fname: (a string)
    Filename from the average subject directory.
    Example: to use rh.entorhinal.label as the input label
    filename, set fname to 'rh.entorhinal' and which to
    'label'. The program will then search for
    '{in_average}/label/rh.entorhinal.label'
    flag: %s, position: -5
hemisphere: ('lh' or 'rh')
    Input hemisphere
    flag: %s, position: -4
in_surf: (an existing file name)
    Input surface file
    flag: %s, position: -3
subject_id: (a string)
    Output subject id
    flag: -o %s
which: ('coords' or 'label' or 'vals' or 'curv' or 'area')
    No documentation
    flag: %s, position: -6

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
```

```

    of class 'str', nipy default value: {})
    Environment variables
erode: (an integer (int or long))
    Undocumented
    flag: -erode %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_average: (a directory name)
    Average subject
    flag: %s, position: -2
in_orig: (an existing file name)
    Original surface filename
    flag: -orig %s
out_file: (a file name)
    Output filename
    flag: %s, position: -1
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (a float)
    Undocumented
    flag: -t %.1f

```

Outputs:

```

out_file: (a file name)
    Output label

```

interfaces.freesurfer.preprocess

ApplyVolTransform

[Link to code](#)Wraps command **mri_vol2vol**

Use FreeSurfer mri_vol2vol to apply a transform.

Examples

```

>>> from nipy.interfaces.freesurfer import ApplyVolTransform
>>> applyreg = ApplyVolTransform()
>>> applyreg.inputs.source_file = 'structural.nii'
>>> applyreg.inputs.reg_file = 'register.dat'
>>> applyreg.inputs.transformed_file = 'struct_warped.nii'
>>> applyreg.inputs.fs_target = True
>>> applyreg.cmdline
'mri_vol2vol --fstarg --reg register.dat --mov structural.nii --o struct_warped.
↪nii'

```

Inputs:

```

[Mandatory]
fs_target: (a boolean)

```

```

        use orig.mgz from subject in regfile as target
        flag: --fstarg
        mutually_exclusive: target_file, tal, fs_target
        requires: reg_file
fsl_reg_file: (an existing file name)
        fslRAS-to-fslRAS matrix (FSL format)
        flag: --fsl %s
        mutually_exclusive: reg_file, fsl_reg_file, xfm_reg_file,
            reg_header, subject
reg_file: (an existing file name)
        tkRAS-to-tkRAS matrix (tkregister2 format)
        flag: --reg %s
        mutually_exclusive: reg_file, fsl_reg_file, xfm_reg_file,
            reg_header, subject
reg_header: (a boolean)
        ScannerRAS-to-ScannerRAS matrix = identity
        flag: --regheader
        mutually_exclusive: reg_file, fsl_reg_file, xfm_reg_file,
            reg_header, subject
source_file: (an existing file name)
        Input volume you wish to transform
        flag: --mov %s
subject: (a unicode string)
        set matrix = identity and use subject for any templates
        flag: --s %s
        mutually_exclusive: reg_file, fsl_reg_file, xfm_reg_file,
            reg_header, subject
tal: (a boolean)
        map to a sub FOV of MNI305 (with --reg only)
        flag: --tal
        mutually_exclusive: target_file, tal, fs_target
target_file: (an existing file name)
        Output template volume
        flag: --targ %s
        mutually_exclusive: target_file, tal, fs_target
xfm_reg_file: (an existing file name)
        ScannerRAS-to-ScannerRAS matrix (MNI format)
        flag: --xfm %s
        mutually_exclusive: reg_file, fsl_reg_file, xfm_reg_file,
            reg_header, subject

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
interp: ('trilin' or 'nearest' or 'cubic')
        Interpolation method (<trilin> or nearest)
        flag: --interp %s
inverse: (a boolean)
        sample from target to source
        flag: --inv

```

```

invert_morph: (a boolean)
    Compute and use the inverse of the non-linear morph to resample the
    input volume. To be used by --m3z.
    flag: --inv-morph
    requires: m3z_file
m3z_file: (a file name)
    This is the morph to be applied to the volume. Unless the morph is
    in mri/transforms (eg.: for talairach.m3z computed by reconall), you
    will need to specify the full path to this morph and use the
    --noDefM3zPath flag.
    flag: --m3z %s
no_ded_m3z_path: (a boolean)
    To be used with the m3z flag. Instructs the code not to look for
    them3z morph in the default location
    (SUBJECTS_DIR/subj/mri/transforms), but instead just use the path
    indicated in --m3z.
    flag: --noDefM3zPath
    requires: m3z_file
no_resample: (a boolean)
    Do not resample; just change vox2ras matrix
    flag: --no-resample
subjects_dir: (an existing directory name)
    subjects directory
tal_resolution: (a float)
    Resolution to sample when using tal
    flag: --talres %.10f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformed_file: (a file name)
    Output volume
    flag: --o %s

```

Outputs:

```

transformed_file: (an existing file name)
    Path to output file if used normally

```

BBRegister[Link to code](#)Wraps command **bbregister**

Use FreeSurfer bregister to register a volume to the Freesurfer anatomical.

This program performs within-subject, cross-modal registration using a boundary-based cost function. It is required that you have an anatomical scan of the subject that has already been recon-all-ed using freesurfer.

Examples

```

>>> from nipy.interfaces.freesurfer import BBRegister
>>> bbreg = BBRegister(subject_id='me', source_file='structural.nii', init='header
↳', contrast_type='t2')
>>> bbreg.cmdline
'bbregister --t2 --init-header --reg structural_bbreg_me.dat --mov structural.nii_
↳--s me'

```

Inputs:

```
[Mandatory]
contrast_type: ('t1' or 't2' or 'bold' or 'dti')
    contrast type of image
    flag: --%s
source_file: (a file name)
    source file to be registered
    flag: --mov %s
subject_id: (a unicode string)
    freesurfer subject id
    flag: --s %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dof: (6 or 9 or 12)
    number of transform degrees of freedom
    flag: --%d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
epi_mask: (a boolean)
    mask out B0 regions in stages 1 and 2
    flag: --epi-mask
fsl_dof: (an integer (int or long))
    degrees of freedom for initial registration (FSL)
    flag: --fsl-dof %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
init: ('coreg' or 'rr' or 'spm' or 'fsl' or 'header' or 'best')
    initialize registration with mri_coreg, spm, fsl, or header
    flag: --init-%s
    mutually_exclusive: init_reg_file
init_reg_file: (an existing file name)
    existing registration file
    flag: --init-reg %s
    mutually_exclusive: init
intermediate_file: (an existing file name)
    Intermediate image, e.g. in case of partial FOV
    flag: --int %s
out_fsl_file: (a boolean or a file name)
    write the transformation matrix in FSL FLIRT format
    flag: --fslmat %s
out_lta_file: (a boolean or a file name)
    write the transformation matrix in LTA format
    flag: --lta %s
out_reg_file: (a file name)
    output registration file
    flag: --reg %s
reg_frame: (an integer (int or long))
    0-based frame index for 4D source file
    flag: --frame %d
    mutually_exclusive: reg_middle_frame
reg_middle_frame: (a boolean)
    Register middle frame of 4D source file
    flag: --mid-frame
```

```

    mutually_exclusive: reg_frame
registered_file: (a boolean or a file name)
    output warped sourcefile either True or filename
    flag: --o %s
spm_nifti: (a boolean)
    force use of nifti rather than analyze with SPM
    flag: --spm-nii
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

min_cost_file: (an existing file name)
    Output registration minimum cost file
out_fsl_file: (a file name)
    Output FLIRT-style registration file
out_lta_file: (a file name)
    Output LTA-style registration file
out_reg_file: (an existing file name)
    Output registration file
registered_file: (a file name)
    Registered and resampled source file

```

CALabel[Link to code](#)Wraps command **mri_ca_label**For complete details, see the [FS Documentation](#)**Examples**

```

>>> from nipy.interfaces import freesurfer
>>> ca_label = freesurfer.CALabel()
>>> ca_label.inputs.in_file = "norm.mgz"
>>> ca_label.inputs.out_file = "out.mgz"
>>> ca_label.inputs.transform = "trans.mat"
>>> ca_label.inputs.template = "Template_6.nii" # in practice use .gcs extension
>>> ca_label.cmdline
'mri_ca_label norm.mgz trans.mat Template_6.nii out.mgz'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    Input volume for CALabel
    flag: %s, position: -4
out_file: (a file name)
    Output file for CALabel
    flag: %s, position: -1
template: (an existing file name)
    Input template for CALabel
    flag: %s, position: -2
transform: (an existing file name)

```

```
    Input transform for CALabel
    flag: %s, position: -3

[Optional]
align: (a boolean)
    Align CALabel
    flag: -align
args: (a unicode string)
    Additional parameters to the command
    flag: %s
aseg: (a file name)
    Undocumented flag. Autorecon3 uses ../mri/aseg.presurf.mgz as input
    file
    flag: -aseg %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_vol: (an existing file name)
    set input volume
    flag: -r %s
intensities: (an existing file name)
    input label intensities file(used in longitudinal processing)
    flag: -r %s
label: (a file name)
    Undocumented flag. Autorecon3 uses
    ../label/{hemisphere}.cortex.label as input file
    flag: -l %s
no_big_ventricles: (a boolean)
    No big ventricles
    flag: -nobigventricles
num_threads: (an integer (int or long))
    allows for specifying more threads
prior: (a float)
    Prior for CALabel
    flag: -prior %.1f
relabel_unlikely: (a tuple of the form: (an integer (int or long), a
    float))
    Reclassify voxels at least some std devs from the mean using some
    size Gaussian window
    flag: -relabel_unlikely %d %.1f
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
    Output volume from CALabel
```


CANormalize

[Link to code](#)

Wraps command `mri_ca_normalize`

This program creates a normalized volume using the brain volume and an input gca file.

For complete details, see the [FS Documentation](#)

Examples

```
>>> from nipy.interfaces import freesurfer
>>> ca_normalize = freesurfer.CANormalize()
>>> ca_normalize.inputs.in_file = "T1.mgz"
>>> ca_normalize.inputs.atlas = "atlas.nii.gz" # in practice use .gca atlases
>>> ca_normalize.inputs.transform = "trans.mat" # in practice use .lta transforms
>>> ca_normalize.cmdline
'mri_ca_normalize T1.mgz atlas.nii.gz trans.mat T1_norm.mgz'
```

Inputs:

```
[Mandatory]
atlas: (an existing file name)
    The atlas file in gca format
    flag: %s, position: -3
in_file: (an existing file name)
    The input file for CANormalize
    flag: %s, position: -4
transform: (an existing file name)
    The tranform file in lta format
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
control_points: (a file name)
    File name for the output control points
    flag: -c %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
long_file: (a file name)
    undocumented flag used in longitudinal processing
    flag: -long %s
mask: (an existing file name)
    Specifies volume to use as mask
    flag: -mask %s
out_file: (a file name)
    The output file for CANormalize
    flag: %s, position: -1
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
```

```
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
control_points: (a file name)
    The output control points for Normalize
out_file: (a file name)
    The output file for Normalize
```

CAResister[Link to code](#)Wraps command **mri_ca_register**

Generates a multi-dimensional talairach transform from a gca file and talairach.lta file

For complete details, see the [FS Documenta**tion**](#)**Examples**

```
>>> from nipy.interfaces import freesurfer
>>> ca_register = freesurfer.CAResister()
>>> ca_register.inputs.in_file = "norm.mgz"
>>> ca_register.inputs.out_file = "talairach.m3z"
>>> ca_register.cmdline
'mri_ca_register norm.mgz talairach.m3z'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    The input volume for CAResister
    flag: %s, position: -3

[Optional]
A: (an integer (int or long))
    undocumented flag used in longitudinal processing
    flag: -A %d
align: (a string)
    Specifies when to perform alignment
    flag: -align-%s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert_and_save: (a boolean)
    Invert and save the .m3z multi-dimensional talaraich transform to x,
    y, and z .mgz files
    flag: -invert-and-save, position: -4
l_files: (a list of items which are a file name)
    undocumented flag used in longitudinal processing
    flag: -l %s
levels: (an integer (int or long))
```

```

        defines how many surrounding voxels will be used in interpolations,
        default is 6
        flag: -levels %d
mask: (an existing file name)
    Specifies volume to use as mask
    flag: -mask %s
no_big_ventricles: (a boolean)
    No big ventricles
    flag: -nobigventricles
num_threads: (an integer (int or long))
    allows for specifying more threads
out_file: (a file name)
    The output volume for CRegister
    flag: %s, position: -1
subjects_dir: (an existing directory name)
    subjects directory
template: (an existing file name)
    The template file in gca format
    flag: %s, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transform: (an existing file name)
    Specifies transform in lta format
    flag: -T %s

```

Outputs:

```

out_file: (a file name)
    The output file for CRegister

```

ConcatenateLTA[Link to code](#)Wraps command **mri_concatenate_lta**concatenates two consecutive LTA transformations into one overall transformation, $Out = LTA2 * LTA1$ **Examples**

```

>>> from nipy.interfaces.freesurfer import ConcatenateLTA
>>> conc_lta = ConcatenateLTA()
>>> conc_lta.inputs.in_lta1 = 'trans.mat'
>>> conc_lta.inputs.in_lta2 = 'trans.mat'
>>> conc_lta.cmdline
'mri_concatenate_lta trans.mat trans.mat trans-long.mat'

```

Inputs:

```

[Mandatory]
in_lta1: (an existing file name)
    maps some src1 to dst1
    flag: %s, position: -3
in_lta2: (an existing file name)
    maps dst1(src2) to dst2
    flag: %s, position: -2

```

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    the combined LTA maps: src1 to dst2 = LTA2*LTA1
    flag: %s, position: -1
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
    the combined LTA maps: src1 to dst2 = LTA2*LTA1
```

DICOMConvert[Link to code](#)Wraps command **mri_convert**

use fs mri_convert to convert dicom files

Examples

```
>>> from nipy.interfaces.freesurfer import DICOMConvert
>>> cvt = DICOMConvert()
>>> cvt.inputs.dicom_dir = 'dicomdir'
>>> cvt.inputs.file_mapping = [('nifti', '*.nii'), ('info', 'dicom*.txt'), ('dti',
↪ '*dti.bv*')]
```

Inputs:

```
[Mandatory]
base_output_dir: (a directory name)
    directory in which subject directories are created
dicom_dir: (an existing directory name)
    dicom directory from which to convert dicom files

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dicom_info: (an existing file name)
    File containing summary information from mri_parse_sdcmdir
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
```

```

    Environment variables
file_mapping: (a list of items which are a tuple of the form: (a
    unicode string, a unicode string))
    defines the output fields of interface
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
ignore_single_slice: (a boolean)
    ignore volumes containing a single slice
    requires: dicom_info
out_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
    'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or
    'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti1' or 'nii' or
    'niigz', nipy default value: niigz)
    defines the type of output file produced
seq_list: (a list of items which are a unicode string)
    list of pulse sequence names to be converted.
    requires: dicom_info
subject_dir_template: (a unicode string, nipy default value:
    S.%04d)
    template for subject directory name
subject_id: (any value)
    subject identifier to insert into template
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

None

EditWMwithAseg

[Link to code](#)

Wraps command `mri_edit_wm_with_aseg`

Edits a wm file using a segmentation

Examples

```

>>> from nipy.interfaces.freesurfer import EditWMwithAseg
>>> editwm = EditWMwithAseg()
>>> editwm.inputs.in_file = "T1.mgz"
>>> editwm.inputs.brain_file = "norm.mgz"
>>> editwm.inputs.seg_file = "aseg.mgz"
>>> editwm.inputs.out_file = "wm.asegedit.mgz"
>>> editwm.inputs.keep_in = True
>>> editwm.cmdline
'mri_edit_wm_with_aseg -keep-in T1.mgz norm.mgz aseg.mgz wm.asegedit.mgz'

```

Inputs:

```

[Mandatory]
brain_file: (an existing file name)
    Input brain/T1 file

```

```

        flag: %s, position: -3
in_file: (an existing file name)
        Input white matter segmentation file
        flag: %s, position: -4
out_file: (a file name)
        File to be written as output
        flag: %s, position: -1
seg_file: (an existing file name)
        Input presurf segmentation file
        flag: %s, position: -2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
keep_in: (a boolean)
        Keep edits as found in input volume
        flag: -keep-in
subjects_dir: (an existing directory name)
        subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
        Output edited WM file

```

FitMSParams[Link to code](#)Wraps command **mri_ms_fitparms**

Estimate tissue paramaters from a set of FLASH images.

Examples

```

>>> from nipy.interfaces.freesurfer import FitMSParams
>>> msfit = FitMSParams()
>>> msfit.inputs.in_files = ['flash_05.mgz', 'flash_30.mgz']
>>> msfit.inputs.out_dir = 'flash_parameters'
>>> msfit.cmdline
'mri_ms_fitparms flash_05.mgz flash_30.mgz flash_parameters'

```

Inputs:

```

[Mandatory]
in_files: (a list of items which are an existing file name)
        list of FLASH images (must be in mgh format)

```

```

        flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
flip_list: (a list of items which are an integer (int or long))
    list of flip angles of the input files
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_dir: (a directory name)
    directory to store output in
    flag: %s, position: -1
subjects_dir: (an existing directory name)
    subjects directory
te_list: (a list of items which are a float)
    list of TEs of the input files (in msec)
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tr_list: (a list of items which are an integer (int or long))
    list of TRs of the input files (in msec)
xfm_list: (a list of items which are an existing file name)
    list of transform files to apply to each FLASH image

```

Outputs:

```

pd_image: (an existing file name)
    image of estimated proton density values
t1_image: (an existing file name)
    image of estimated T1 relaxation values
t2star_image: (an existing file name)
    image of estimated T2* values

```

MNIBiasCorrection[Link to code](#)Wraps command **mri_nu_correct.mni**

Wrapper for `nu_correct`, a program from the Montreal Neurological Insitute (MNI) used for correcting intensity non-uniformity (ie, bias fields). You must have the MNI software installed on your system to run this. See [\[www.bic.mni.mcgill.ca/software/N3\]](http://www.bic.mni.mcgill.ca/software/N3) for more info.

`mri_nu_correct.mni` uses float internally instead of uchar. It also rescales the output so that the global mean is the same as that of the input. These two changes are linked and can be turned off with `-no-float`

Examples

```

>>> from nipy.interfaces.freesurfer import MNIBiasCorrection
>>> correct = MNIBiasCorrection()
>>> correct.inputs.in_file = "norm.mgz"
>>> correct.inputs.iterations = 6

```

```
>>> correct.inputs.protocol_iterations = 1000
>>> correct.inputs.distance = 50
>>> correct.cmdline
'mri_nu_correct.mni --distance 50 --i norm.mgz --n 6 --o norm_output.mgz --proto-
↪iters 1000'
```

References:

[http://freesurfer.net/fswiki/mri_nu_correct.mni] [<http://www.bic.mni.mcgill.ca/software/N3>] [<https://github.com/BIC-MNI/N3>]

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input volume. Input can be any format accepted by mri_convert.
        flag: --i %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
distance: (an integer (int or long))
          N3 -distance option
          flag: --distance %d
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
iterations: (an integer (int or long))
            Number of iterations to run nu_correct. Default is 4. This is the
            number of times that nu_correct is repeated (ie, using the output
            from the previous run as the input for the next). This is different
            than the -iterations option to nu_correct.
            flag: --n %d
mask: (an existing file name)
      brainmask volume. Input can be any format accepted by mri_convert.
      flag: --mask %s
no_rescale: (a boolean)
            do not rescale so that global mean of output == input global mean
            flag: --no-rescale
out_file: (a file name)
          output volume. Output can be any format accepted by mri_convert. If
          the output format is COR, then the directory must exist.
          flag: --o %s
protocol_iterations: (an integer (int or long))
                    Passes Np as argument of the -iterations flag of nu_correct. This is
                    different than the --n flag above. Default is not to pass nu_correct
                    the -iterations flag.
                    flag: --proto-iters %d
shrink: (an integer (int or long))
        Shrink parameter for finer sampling (default is 4)
        flag: --shrink %d
stop: (a float)
      Convergence threshold below which iteration stops (suggest 0.01 to
```



```

0.0001)
flag: --stop %f
subjects_dir: (an existing directory name)
               subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
                        (default), `allatonce` - waits till command is finished to display
                        output, `file` - writes output to file, `none` - output is ignored
transform: (an existing file name)
            tal.xfm. Use mri_make_uchar instead of conforming
flag: --uchar %s

```

Outputs:

```

out_file: (an existing file name)
          output volume

```

MRIConvert[Link to code](#)Wraps command **mri_convert**

use fs mri_convert to manipulate files

Note: Adds niigz as an output type option**Examples**

```

>>> mc = MRIConvert()
>>> mc.inputs.in_file = 'structural.nii'
>>> mc.inputs.out_file = 'outfile.mgz'
>>> mc.inputs.out_type = 'mgz'
>>> mc.cmdline
'mri_convert --out_type mgz --input_volume structural.nii --output_volume outfile.
↪mgz'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         File to read/convert
         flag: --input_volume %s, position: -2

[Optional]
apply_inv_transform: (an existing file name)
                    apply inverse transformation xfm file
                    flag: --apply_inverse_transform %s
apply_transform: (an existing file name)
                 apply xfm file
                 flag: --apply_transform %s
args: (a unicode string)
      Additional parameters to the command
      flag: %s
ascii: (a boolean)
       save output as ascii col>row>slice>frame
       flag: --ascii
autoalign_matrix: (an existing file name)

```

```

        text file with autoalign matrix
        flag: --autoalign %s
color_file: (an existing file name)
        color file
        flag: --color_file %s
conform: (a boolean)
        conform to 1mm voxel size in coronal slice direction with 256^3 or
        more
        flag: --conform
conform_min: (a boolean)
        conform to smallest size
        flag: --conform_min
conform_size: (a float)
        conform to size_in_mm
        flag: --conform_size %s
crop_center: (a tuple of the form: (an integer (int or long), an
        integer (int or long), an integer (int or long)))
        <x> <y> <z> crop to 256 around center (x, y, z)
        flag: --crop %d %d %d
crop_gdf: (a boolean)
        apply GDF cropping
        flag: --crop_gdf
crop_size: (a tuple of the form: (an integer (int or long), an
        integer (int or long), an integer (int or long)))
        <dx> <dy> <dz> crop to size <dx, dy, dz>
        flag: --cropsizes %d %d %d
cut_ends: (an integer (int or long))
        remove ncut slices from the ends
        flag: --cutends %d
cw256: (a boolean)
        conform to dimensions of 256^3
        flag: --cw256
devolve_transform: (a unicode string)
        subject id
        flag: --devolve_xfm %s
drop_n: (an integer (int or long))
        drop the last n frames
        flag: --ndrop %d
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
fill_parcellation: (a boolean)
        fill parcellation
        flag: --fill_parcellation
force_ras: (a boolean)
        use default when orientation info absent
        flag: --force_ras_good
frame: (an integer (int or long))
        keep only 0-based frame number
        flag: --frame %d
frame_subsample: (a tuple of the form: (an integer (int or long), an
        integer (int or long), an integer (int or long)))
        start delta end : frame subsampling (end = -1 for end)
        flag: --fsubsample %d %d %d
fwhm: (a float)
        smooth input volume by fwhm mm
        flag: --fwhm %f

```

```

ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_center: (a list of at most 3 items which are a float)
    <R coordinate> <A coordinate> <S coordinate>
    flag: --in_center %s
in_i_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    flag: --in_i_direction %f %f %f
in_i_size: (an integer (int or long))
    input i size
    flag: --in_i_size %d
in_info: (a boolean)
    display input info
    flag: --in_info
in_j_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    flag: --in_j_direction %f %f %f
in_j_size: (an integer (int or long))
    input j size
    flag: --in_j_size %d
in_k_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    flag: --in_k_direction %f %f %f
in_k_size: (an integer (int or long))
    input k size
    flag: --in_k_size %d
in_like: (an existing file name)
    input looks like
    flag: --in_like %s
in_matrix: (a boolean)
    display input matrix
    flag: --in_matrix
in_orientation: ('LAI' or 'LIA' or 'ALI' or 'AIL' or 'ILA' or 'IAL'
    or 'LAS' or 'LSA' or 'ALS' or 'ASL' or 'SLA' or 'SAL' or 'LPI' or
    'LIP' or 'PLI' or 'PIL' or 'ILP' or 'IPL' or 'LPS' or 'LSP' or
    'PLS' or 'PSL' or 'SLP' or 'SPL' or 'RAI' or 'RIA' or 'ARI' or
    'AIR' or 'IRA' or 'IAR' or 'RAS' or 'RSA' or 'ARS' or 'ASR' or
    'SRA' or 'SAR' or 'RPI' or 'RIP' or 'PRI' or 'PIR' or 'IRP' or
    'IPR' or 'RPS' or 'RSP' or 'PRS' or 'PSR' or 'SRP' or 'SPR')
    specify the input orientation
    flag: --in_orientation %s
in_scale: (a float)
    input intensity scale factor
    flag: --scale %f
in_stats: (a boolean)
    display input stats
    flag: --in_stats
in_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
    'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or
    'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti1' or 'nii' or
    'niigz' or 'ge' or 'gelx' or 'lx' or 'ximg' or 'siemens' or 'dicom'
    or 'siemens_dicom')
    input file type
    flag: --in_type %s
invert_contrast: (a float)
    threshold for inversting contrast
    flag: --invert_contrast %f

```

```
midframe: (a boolean)
    keep only the middle frame
    flag: --mid-frame
no_change: (a boolean)
    don't change type of input to that of template
    flag: --nochange
no_scale: (a boolean)
    dont rescale values for COR
    flag: --no_scale 1
no_translate: (a boolean)
    ~~~
    flag: --no_translate
no_write: (a boolean)
    do not write output
    flag: --no_write
out_center: (a tuple of the form: (a float, a float, a float))
    <R coordinate> <A coordinate> <S coordinate>
    flag: --out_center %f %f %f
out_datatype: ('uchar' or 'short' or 'int' or 'float')
    output data type <uchar|short|int|float>
    flag: --out_data_type %s
out_file: (a file name)
    output filename or True to generate one
    flag: --output_volume %s, position: -1
out_i_count: (an integer (int or long))
    some count ?? in i direction
    flag: --out_i_count %d
out_i_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    flag: --out_i_direction %f %f %f
out_i_size: (an integer (int or long))
    output i size
    flag: --out_i_size %d
out_info: (a boolean)
    display output info
    flag: --out_info
out_j_count: (an integer (int or long))
    some count ?? in j direction
    flag: --out_j_count %d
out_j_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    flag: --out_j_direction %f %f %f
out_j_size: (an integer (int or long))
    output j size
    flag: --out_j_size %d
out_k_count: (an integer (int or long))
    some count ?? in k direction
    flag: --out_k_count %d
out_k_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    flag: --out_k_direction %f %f %f
out_k_size: (an integer (int or long))
    output k size
    flag: --out_k_size %d
out_matrix: (a boolean)
    display output matrix
    flag: --out_matrix
out_orientation: ('LAI' or 'LIA' or 'ALI' or 'AIL' or 'ILA' or 'IAL')
```

```

    or 'LAS' or 'LSA' or 'ALS' or 'ASL' or 'SLA' or 'SAL' or 'LPI' or
    'LIP' or 'PLI' or 'PIL' or 'ILP' or 'IPL' or 'LPS' or 'LSP' or
    'PLS' or 'PSL' or 'SLP' or 'SPL' or 'RAI' or 'RIA' or 'ARI' or
    'AIR' or 'IRA' or 'IAR' or 'RAS' or 'RSA' or 'ARS' or 'ASR' or
    'SRA' or 'SAR' or 'RPI' or 'RIP' or 'PRI' or 'PIR' or 'IRP' or
    'IPR' or 'RPS' or 'RSP' or 'PRS' or 'PSR' or 'SRP' or 'SPR')
    specify the output orientation
    flag: --out_orientation %s
out_scale: (a float)
    output intensity scale factor
    flag: --out-scale %d
out_stats: (a boolean)
    display output stats
    flag: --out_stats
out_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
    'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or
    'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti1' or 'nii' or
    'niigz')
    output file type
    flag: --out_type %s
parse_only: (a boolean)
    parse input only
    flag: --parse_only
read_only: (a boolean)
    read the input volume
    flag: --read_only
reorder: (a tuple of the form: (an integer (int or long), an integer
    (int or long), an integer (int or long)))
    olddim1 olddim2 olddim3
    flag: --reorder %d %d %d
resample_type: ('interpolate' or 'weighted' or 'nearest' or 'sinc' or
    'cubic')
    <interpolate|weighted|nearest|sinc|cubic> (default is interpolate)
    flag: --resample_type %s
reslice_like: (an existing file name)
    reslice output to match file
    flag: --reslice_like %s
sdcmlist: (an existing file name)
    list of DICOM files for conversion
    flag: --sdcmlist %s
skip_n: (an integer (int or long))
    skip the first n frames
    flag: --nskip %d
slice_bias: (a float)
    apply half-cosine bias field
    flag: --slice-bias %f
slice_crop: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    s_start s_end : keep slices s_start to s_end
    flag: --slice-crop %d %d
slice_reverse: (a boolean)
    reverse order of slices, update vox2ras
    flag: --slice-reverse
smooth_parcellation: (a boolean)
    smooth parcellation
    flag: --smooth_parcellation
sphinx: (a boolean)
    change orientation info to sphinx

```

```

    flag: --sphinx
split: (a boolean)
    split output frames into separate output files.
    flag: --split
status_file: (a file name)
    status file for DICOM conversion
    flag: --status %s
subject_name: (a unicode string)
    subject name ???
    flag: --subject_name %s
subjects_dir: (an existing directory name)
    subjects directory
te: (an integer (int or long))
    TE in msec
    flag: -te %d
template_info: (a boolean)
    dump info about template
template_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
    'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or
    'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'nii' or
    'niigz' or 'ge' or 'gelx' or 'lx' or 'ximg' or 'siemens' or 'dicom'
    or 'siemens_dicom')
    template file type
    flag: --template_type %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
ti: (an integer (int or long))
    TI in msec (note upper case flag)
    flag: -ti %d
tr: (an integer (int or long))
    TR in msec
    flag: -tr %d
unwarp_gradient: (a boolean)
    unwarp gradient nonlinearity
    flag: --unwarp_gradient_nonlinearity
vox_size: (a tuple of the form: (a float, a float, a float))
    <size_x> <size_y> <size_z> specify the size (mm) - useful for
    upsampling or downsampling
    flag: -voxsize %f %f %f
zero_ge_z_offset: (a boolean)
    zero ge z offset ???
    flag: --zero_ge_z_offset
zero_outlines: (a boolean)
    zero outlines
    flag: --zero_outlines

```

Outputs:

```

out_file: (a list of items which are an existing file name)
    converted output file

```

MRIsCAlabel[Link to code](#)Wraps command `mris_ca_label`

For a single subject, produces an annotation file, in which each cortical surface vertex is assigned a neuroanatom-

ical label. This automatic procedure employs data from a previously-prepared atlas file. An atlas file is created from a training set, capturing region data manually drawn by neuroanatomists combined with statistics on variability correlated to geometric information derived from the cortical model (sulcus and curvature). Besides the atlases provided with FreeSurfer, new ones can be prepared using `mrisc_a_train`).

Examples

```
>>> from nipy.interfaces import freesurfer
>>> ca_label = freesurfer.MRIsCALabel()
>>> ca_label.inputs.subject_id = "test"
>>> ca_label.inputs.hemisphere = "lh"
>>> ca_label.inputs.canonsurf = "lh.pial"
>>> ca_label.inputs.curv = "lh.pial"
>>> ca_label.inputs.sulc = "lh.pial"
>>> ca_label.inputs.classifier = "iml.nii" # in practice, use .gcs extension
>>> ca_label.inputs.smoothwm = "lh.pial"
>>> ca_label.cmdline
'mrisc_a_label test lh lh.pial iml.nii lh.aparc.annot'
```

Inputs:

```
[Mandatory]
canonsurf: (an existing file name)
    Input canonical surface file
    flag: %s, position: -3
classifier: (an existing file name)
    Classifier array input file
    flag: %s, position: -2
curv: (an existing file name)
    implicit input {hemisphere}.curv
hemisphere: ('lh' or 'rh')
    Hemisphere ('lh' or 'rh')
    flag: %s, position: -4
smoothwm: (an existing file name)
    implicit input {hemisphere}.smoothwm
subject_id: (a string, nipy default value: subject_id)
    Subject name or ID
    flag: %s, position: -5
sulc: (an existing file name)
    implicit input {hemisphere}.sulc

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
aseg: (a file name)
    Undocumented flag. Autorecon3 uses ../mri/aseg.presurf.mgz as input
    file
    flag: -aseg %s
copy_inputs: (a boolean)
    Copies implicit inputs to node directory and creates a temp
    subjects_directory. Use this when running as a node
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
```

```

        interface fails to run
label: (a file name)
    Undocumented flag. Autorecon3 uses
    ../label/{hemisphere}.cortex.label as input file
    flag: -l %s
num_threads: (an integer (int or long))
    allows for specifying more threads
out_file: (a file name)
    Annotated surface output file
    flag: %s, position: -1
seed: (an integer (int or long))
    flag: -seed %d
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
    Output volume from MRIsCLabel

```

Normalize[Link to code](#)Wraps command **mri_normalize**

Normalize the white-matter, optionally based on control points. The input volume is converted into a new volume where white matter image values all range around 110.

Examples

```

>>> from nipy.interfaces import freesurfer
>>> normalize = freesurfer.Normalize()
>>> normalize.inputs.in_file = "T1.mgz"
>>> normalize.inputs.gradient = 1
>>> normalize.cmdline
'mri_normalize -g 1 T1.mgz T1_norm.mgz'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    The input file for Normalize
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gradient: (an integer (int or long))
    use max intensity/mm gradient g (default=1)

```



```

    flag: -g %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    The input mask file for Normalize
    flag: -mask %s
out_file: (a file name)
    The output file for Normalize
    flag: %s, position: -1
segmentation: (an existing file name)
    The input segmentation for Normalize
    flag: -aseg %s
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transform: (an existing file name)
    Tranform file from the header of the input file

```

Outputs:

```

out_file: (a file name)
    The output file for Normalize

```

ParseDICOMDir[Link to code](#)Wraps command **mri_parse_sdcmdir**

Uses mri_parse_sdcmdir to get information from dicom directories

Examples

```

>>> from nipy.interfaces.freesurfer import ParseDICOMDir
>>> dcminfo = ParseDICOMDir()
>>> dcminfo.inputs.dicom_dir = '.'
>>> dcminfo.inputs.sortbyrun = True
>>> dcminfo.inputs.summarize = True
>>> dcminfo.cmdline
'mri_parse_sdcmdir --d . --o dicominfo.txt --sortbyrun --summarize'

```

Inputs:

```

[Mandatory]
dicom_dir: (an existing directory name)
    path to siemens dicom directory
    flag: --d %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dicom_info_file: (a file name, nipy default value: dicominfo.txt)
    file to which results are written
    flag: --o %s

```

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
         Print an error message instead of throwing an exception in case the
         interface fails to run
sortbyrun: (a boolean)
         assign run numbers
         flag: --sortbyrun
subjects_dir: (an existing directory name)
         subjects directory
summarize: (a boolean)
         only print out info for run leaders
         flag: --summarize
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
         Control terminal output: `stream` - displays to terminal immediately
         (default), `allatonce` - waits till command is finished to display
         output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

dicom_info_file: (an existing file name)
                 text file containing dicom information

```

ReconAll[Link to code](#)Wraps command **recon-all**

Uses recon-all to generate surfaces and parcellations of structural data from anatomical images of a subject.

Examples

```

>>> from nipy.interfaces.freesurfer import ReconAll
>>> reconall = ReconAll()
>>> reconall.inputs.subject_id = 'foo'
>>> reconall.inputs.directive = 'all'
>>> reconall.inputs.subjects_dir = '.'
>>> reconall.inputs.T1_files = 'structural.nii'
>>> reconall.cmdline
'recon-all -all -i structural.nii -subjid foo -sd .'
>>> reconall.inputs.flags = "-qcache"
>>> reconall.cmdline
'recon-all -all -i structural.nii -qcache -subjid foo -sd .'
>>> reconall.inputs.flags = ["-cw256", "-qcache"]
>>> reconall.cmdline
'recon-all -all -i structural.nii -cw256 -qcache -subjid foo -sd .'

```

Hemisphere may be specified regardless of directive:

```

>>> reconall.inputs.flags = []
>>> reconall.inputs.hemi = 'lh'
>>> reconall.cmdline
'recon-all -all -i structural.nii -hemi lh -subjid foo -sd .'

```

-autorecon-hemi uses the **-hemi** input to specify the hemisphere to operate upon:

```
>>> reconall.inputs.directive = 'autorecon-hemi'
>>> reconall.cmdline
'recon-all -autorecon-hemi lh -i structural.nii -subjid foo -sd .'
```

Hippocampal subfields can accept T1 and T2 images:

```
>>> reconall_subfields = ReconAll()
>>> reconall_subfields.inputs.subject_id = 'foo'
>>> reconall_subfields.inputs.directive = 'all'
>>> reconall_subfields.inputs.subjects_dir = '.'
>>> reconall_subfields.inputs.T1_files = 'structural.nii'
>>> reconall_subfields.inputs.hippocampal_subfields_T1 = True
>>> reconall_subfields.cmdline
'recon-all -all -i structural.nii -hippocampal-subfields-T1 -subjid foo -sd .'
>>> reconall_subfields.inputs.hippocampal_subfields_T2 = (
... 'structural.nii', 'test')
>>> reconall_subfields.cmdline
'recon-all -all -i structural.nii -hippocampal-subfields-T1T2 structural.nii test_
↳-subjid foo -sd .'
>>> reconall_subfields.inputs.hippocampal_subfields_T1 = False
>>> reconall_subfields.cmdline
'recon-all -all -i structural.nii -hippocampal-subfields-T2 structural.nii test -
↳-subjid foo -sd .'
```

Inputs:

```
[Mandatory]

[Optional]
T1_files: (a list of items which are an existing file name)
    name of T1 file to process
    flag: -i %s...
T2_file: (an existing file name)
    Convert T2 image to orig directory
    flag: -T2 %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
big_ventricles: (a boolean)
    For use in subjects with enlarged ventricles
    flag: -bigventricles
brainstem: (a boolean)
    Segment brainstem structures
    flag: -brainstem-structures
directive: ('all' or 'autorecon1' or 'autorecon2' or
    'autorecon2-volonly' or 'autorecon2-perhemi' or
    'autorecon2-inflate1' or 'autorecon2-cp' or 'autorecon2-wm' or
    'autorecon3' or 'autorecon3-T2pial' or 'autorecon-pial' or
    'autorecon-hemi' or 'localGI' or 'qcache', nipype default value:
    all)
    process directive
    flag: -%s, position: 0
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
expert: (an existing file name)
    Set parameters using expert file
    flag: -expert %s
```

```
flags: (a list of items which are a unicode string)
    additional parameters
    flag: %s
hemi: ('lh' or 'rh')
    hemisphere to process
    flag: -hemi %s
hippocampal_subfields_T1: (a boolean)
    segment hippocampal subfields using input T1 scan
    flag: -hippocampal-subfields-T1
hippocampal_subfields_T2: (a tuple of the form: (an existing file
    name, a unicode string))
    segment hippocampal subfields using T2 scan, identified by ID (may
    be combined with hippocampal_subfields_T1)
    flag: -hippocampal-subfields-T2 %s %s
hires: (a boolean)
    Conform to minimum voxel size (for voxels < 1mm)
    flag: -hires
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mprage: (a boolean)
    Assume scan parameters are MGH MP-RAGE protocol, which produces
    darker gray matter
    flag: -mprage
mri_aparc2aseg: (a unicode string)
    Flags to pass to mri_aparc2aseg commands
    mutually_exclusive: expert
mri_ca_label: (a unicode string)
    Flags to pass to mri_ca_label commands
    mutually_exclusive: expert
mri_ca_normalize: (a unicode string)
    Flags to pass to mri_ca_normalize commands
    mutually_exclusive: expert
mri_ca_register: (a unicode string)
    Flags to pass to mri_ca_register commands
    mutually_exclusive: expert
mri_edit_wm_with_aseg: (a unicode string)
    Flags to pass to mri_edit_wm_with_aseg commands
    mutually_exclusive: expert
mri_em_register: (a unicode string)
    Flags to pass to mri_em_register commands
    mutually_exclusive: expert
mri_fill: (a unicode string)
    Flags to pass to mri_fill commands
    mutually_exclusive: expert
mri_mask: (a unicode string)
    Flags to pass to mri_mask commands
    mutually_exclusive: expert
mri_normalize: (a unicode string)
    Flags to pass to mri_normalize commands
    mutually_exclusive: expert
mri_preteess: (a unicode string)
    Flags to pass to mri_preteess commands
    mutually_exclusive: expert
mri_remove_neck: (a unicode string)
    Flags to pass to mri_remove_neck commands
    mutually_exclusive: expert
mri_segment: (a unicode string)
```

```

        Flags to pass to mri_segment commands
        mutually_exclusive: expert
mri_segstats: (a unicode string)
        Flags to pass to mri_segstats commands
        mutually_exclusive: expert
mri_tessellate: (a unicode string)
        Flags to pass to mri_tessellate commands
        mutually_exclusive: expert
mri_watershed: (a unicode string)
        Flags to pass to mri_watershed commands
        mutually_exclusive: expert
mris_anatomical_stats: (a unicode string)
        Flags to pass to mris_anatomical_stats commands
        mutually_exclusive: expert
mris_ca_label: (a unicode string)
        Flags to pass to mris_ca_label commands
        mutually_exclusive: expert
mris_fix_topology: (a unicode string)
        Flags to pass to mris_fix_topology commands
        mutually_exclusive: expert
mris_inflate: (a unicode string)
        Flags to pass to mri_inflate commands
        mutually_exclusive: expert
mris_make_surfaces: (a unicode string)
        Flags to pass to mris_make_surfaces commands
        mutually_exclusive: expert
mris_register: (a unicode string)
        Flags to pass to mris_register commands
        mutually_exclusive: expert
mris_smooth: (a unicode string)
        Flags to pass to mri_smooth commands
        mutually_exclusive: expert
mris_sphere: (a unicode string)
        Flags to pass to mris_sphere commands
        mutually_exclusive: expert
mris_surf2vol: (a unicode string)
        Flags to pass to mris_surf2vol commands
        mutually_exclusive: expert
mrisc_paint: (a unicode string)
        Flags to pass to mrisc_paint commands
        mutually_exclusive: expert
openmp: (an integer (int or long))
        Number of processors to use in parallel
        flag: -openmp %d
parallel: (a boolean)
        Enable parallel execution
        flag: -parallel
subject_id: (a unicode string, nipy default value: recon_all)
        subject name
        flag: -subjid %s
subjects_dir: (an existing directory name)
        path to subjects directory
        flag: -sd %s
talairach: (a unicode string)
        Flags to pass to talairach commands
        mutually_exclusive: expert
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately

```

```

        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
use_T2: (a boolean)
        Use converted T2 to refine the cortical surface
        flag: -T2pial
xopts: ('use' or 'clean' or 'overwrite')
        Use, delete or overwrite existing expert options file
        flag: -xopts-%s

```

Outputs:

```

BA_stats: (a list of items which are an existing file name)
        Brodmann Area statistics files
T1: (an existing file name)
        Intensity normalized whole-head volume
annot: (a list of items which are an existing file name)
        Surface annotation files
aparc_a2009s_stats: (a list of items which are an existing file name)
        Aparc a2009s parcellation statistics files
aparc_aseg: (a list of items which are an existing file name)
        Aparc parcellation projected into aseg volume
aparc_stats: (a list of items which are an existing file name)
        Aparc parcellation statistics files
area_pial: (a list of items which are an existing file name)
        Mean area of triangles each vertex on the pial surface is associated
        with
aseg: (an existing file name)
        Volumetric map of regions from automatic segmentation
aseg_stats: (a list of items which are an existing file name)
        Automated segmentation statistics file
avg_curv: (a list of items which are an existing file name)
        Average atlas curvature, sampled to subject
brain: (an existing file name)
        Intensity normalized brain-only volume
brainmask: (an existing file name)
        Skull-stripped (brain-only) volume
curv: (a list of items which are an existing file name)
        Maps of surface curvature
curv_pial: (a list of items which are an existing file name)
        Curvature of pial surface
curv_stats: (a list of items which are an existing file name)
        Curvature statistics files
entorhinal_exvivo_stats: (a list of items which are an existing file
        name)
        Entorhinal exvivo statistics files
filled: (an existing file name)
        Subcortical mass volume
graymid: (a list of items which are an existing file name)
        Graymid/midthickness surface meshes
inflated: (a list of items which are an existing file name)
        Inflated surface meshes
jacobian_white: (a list of items which are an existing file name)
        Distortion required to register to spherical atlas
label: (a list of items which are an existing file name)
        Volume and surface label files
norm: (an existing file name)
        Normalized skull-stripped volume
nu: (an existing file name)

```

```

    Non-uniformity corrected whole-head volume
orig: (an existing file name)
    Base image conformed to Freesurfer space
pial: (a list of items which are an existing file name)
    Gray matter/pia mater surface meshes
rawavg: (an existing file name)
    Volume formed by averaging input images
ribbon: (a list of items which are an existing file name)
    Volumetric maps of cortical ribbons
smoothwm: (a list of items which are an existing file name)
    Smoothed original surface meshes
sphere: (a list of items which are an existing file name)
    Spherical surface meshes
sphere_reg: (a list of items which are an existing file name)
    Spherical registration file
subject_id: (a unicode string)
    Subject name for whom to retrieve data
subjects_dir: (an existing directory name)
    Freesurfer subjects directory.
sulc: (a list of items which are an existing file name)
    Surface maps of sulcal depth
thickness: (a list of items which are an existing file name)
    Surface maps of cortical thickness
volume: (a list of items which are an existing file name)
    Surface maps of cortical volume
white: (a list of items which are an existing file name)
    White/gray matter surface meshes
wm: (an existing file name)
    Segmented white-matter volume
wmparc: (an existing file name)
    Aparc parcellation projected into subcortical white matter
wmparc_stats: (a list of items which are an existing file name)
    White matter parcellation statistics file

```

Resample

[Link to code](#)

Wraps command **mri_convert**

Use FreeSurfer **mri_convert** to up or down-sample image files

Examples

```

>>> from nipy.interfaces import freesurfer
>>> resampler = freesurfer.Resample()
>>> resampler.inputs.in_file = 'structural.nii'
>>> resampler.inputs.resampled_file = 'resampled.nii'
>>> resampler.inputs.voxel_size = (2.1, 2.1, 2.1)
>>> resampler.cmdline
'mri_convert -vs 2.10 2.10 2.10 -i structural.nii -o resampled.nii'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    file to resample
    flag: -i %s, position: -2
voxel_size: (a tuple of the form: (a float, a float, a float))

```

```

        triplet of output voxel sizes
        flag: -vs %.2f %.2f %.2f

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
resampled_file: (a file name)
    output filename
    flag: -o %s, position: -1
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

resampled_file: (an existing file name)
    output filename

```

RobustRegister[Link to code](#)Wraps command **mri_robust_register**

Perform intramodal linear registration (translation and rotation) using robust statistics.

Examples

```

>>> from nipy.interfaces.freesurfer import RobustRegister
>>> reg = RobustRegister()
>>> reg.inputs.source_file = 'structural.nii'
>>> reg.inputs.target_file = 'T1.nii'
>>> reg.inputs.auto_sens = True
>>> reg.inputs.init_orient = True
>>> reg.cmdline
'mri_robust_register --satit --initorient --lta structural_robustreg.lta --mov_
->structural.nii --dst T1.nii'

```

References

Reuter, M, Rosas, HD, and Fischl, B, (2010). Highly Accurate Inverse Consistent Registration: A Robust Approach. Neuroimage 53(4) 1181-96.

Inputs:

```

[Mandatory]
auto_sens: (a boolean)
    auto-detect good sensitivity

```



```

        flag: --satit
        mutually_exclusive: outlier_sens
outlier_sens: (a float)
    set outlier sensitivity explicitly
    flag: --sat %.4f
    mutually_exclusive: auto_sens
source_file: (a file name)
    volume to be registered
    flag: --mov %s
target_file: (a file name)
    target volume for the registration
    flag: --dst %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
est_int_scale: (a boolean)
    estimate intensity scale (recommended for unnormalized images)
    flag: --iscale
force_double: (a boolean)
    use double-precision intensities
    flag: --doubleprec
force_float: (a boolean)
    use float intensities
    flag: --floattype
half_source: (a boolean or a file name)
    write source volume mapped to halfway space
    flag: --halfmov %s
half_source_xfm: (a boolean or a file name)
    write transform from source to halfway space
    flag: --halfmovlta %s
half_targ: (a boolean or a file name)
    write target volume mapped to halfway space
    flag: --halfdst %s
half_targ_xfm: (a boolean or a file name)
    write transform from target to halfway space
    flag: --halfdstlta %s
half_weights: (a boolean or a file name)
    write weights volume mapped to halfway space
    flag: --halfweights %s
high_iterations: (an integer (int or long))
    max # of times on highest resolution
    flag: --highit %d
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_xfm_file: (an existing file name)
    use initial transform on source
    flag: --transform
init_orient: (a boolean)
    use moments for initial orient (recommended for stripped brains)
    flag: --initorient
iteration_thresh: (a float)

```

```
    stop iterations when below threshold
    flag: --epsit %.3f
least_squares: (a boolean)
    use least squares instead of robust estimator
    flag: --leastquares
mask_source: (an existing file name)
    image to mask source volume with
    flag: --maskmov %s
mask_target: (an existing file name)
    image to mask target volume with
    flag: --maskdst %s
max_iterations: (an integer (int or long))
    maximum # of times on each resolution
    flag: --maxit %d
no_init: (a boolean)
    skip transform init
    flag: --noinit
no_multi: (a boolean)
    work on highest resolution
    flag: --nomulti
out_reg_file: (a file name)
    registration file to write
    flag: --lta %s
outlier_limit: (a float)
    set maximal outlier limit in satit
    flag: --wlimit %.3f
registered_file: (a boolean or a file name)
    registered image; either True or filename
    flag: --warp %s
subjects_dir: (an existing directory name)
    subjects directory
subsample_thresh: (an integer (int or long))
    subsample if dimension is above threshold size
    flag: --subsample %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
trans_only: (a boolean)
    find 3 parameter translation only
    flag: --transonly
weights_file: (a boolean or a file name)
    weights image to write; either True or filename
    flag: --weights %s
write_vo2vox: (a boolean)
    output vox2vox matrix (default is RAS2RAS)
    flag: --vox2vox
```

Outputs:

```
half_source: (a file name)
    source image mapped to halfway space
half_source_xfm: (a file name)
    transform file to map source image to halfway space
half_targ: (a file name)
    target image mapped to halfway space
half_targ_xfm: (a file name)
    transform file to map target image to halfway space
```

```

half_weights: (a file name)
    weights image mapped to halfway space
out_reg_file: (an existing file name)
    output registration file
registered_file: (a file name)
    output image with registration applied
weights_file: (a file name)
    image of weights used

```

SegmentCC

[Link to code](#)

Wraps command **mri_cc**

This program segments the corpus callosum into five separate labels in the subcortical segmentation volume 'aseg.mgz'. The divisions of the cc are equally spaced in terms of distance along the primary eigendirection (pretty much the long axis) of the cc. The lateral extent can be changed with the -T <thickness> parameter, where <thickness> is the distance off the midline (so -T 1 would result in the who CC being 3mm thick). The default is 2 so it's 5mm thick. The aseg.stats values should be volume.

Examples

```

>>> from nipy.interfaces import freesurfer
>>> SegmentCC_node = freesurfer.SegmentCC()
>>> SegmentCC_node.inputs.in_file = "aseg.mgz"
>>> SegmentCC_node.inputs.in_norm = "norm.mgz"
>>> SegmentCC_node.inputs.out_rotation = "cc.lta"
>>> SegmentCC_node.inputs.subject_id = "test"
>>> SegmentCC_node.cmdline
'mri_cc -aseg aseg.mgz -o aseg.auto.mgz -lta cc.lta test'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    Input aseg file to read from subjects directory
    flag: -aseg %s
in_norm: (an existing file name)
    Required undocumented input {subject}/mri/norm.mgz
out_rotation: (a file name)
    Global filepath for writing rotation lta
    flag: -lta %s
subject_id: (a string, nipy default value: subject_id)
    Subject name
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the input
    files to the node directory.
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```

```
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    Filename to write aseg including CC
    flag: -o %s
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
    Output segmentation uncluding corpus collosum
out_rotation: (a file name)
    Output lta rotation file
```

SegmentWM

[Link to code](#)

Wraps command **mri_segment**

This program segments white matter from the input volume. The input volume should be normalized such that white matter voxels are ~110-valued, and the volume is conformed to 256^3.

Examples

```
>>> from nipy.interfaces import freesurfer
>>> SegmentWM_node = freesurfer.SegmentWM()
>>> SegmentWM_node.inputs.in_file = "norm.mgz"
>>> SegmentWM_node.inputs.out_file = "wm(seg.mgz"
>>> SegmentWM_node.cmdline
'mri_segment norm.mgz wm(seg.mgz'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    Input file for SegmentWM
    flag: %s, position: -2
out_file: (a file name)
    File to be written as output for SegmentWM
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```



```
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {}))
Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
proj_frac: (a float)
    project frac of thickness a long surface normal
    flag: --projfrac %s
    mutually_exclusive: proj_frac_avg
proj_frac_avg: (a tuple of the form: (a float, a float, a float))
    average a long normal min max delta
    flag: --projfrac-avg %.2f %.2f %.2f
    mutually_exclusive: proj_frac
smoothed_file: (a file name)
    output volume
    flag: --o %s
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
vol_fwhm: (a floating point number >= 0.0)
    volume smoothing outside of surface
    flag: --vol-fwhm %f
```

Outputs:

```
smoothed_file: (an existing file name)
    smoothed input volume
```

SynthesizeFLASH

[Link to code](#)

Wraps command **mri_synthesize**

Synthesize a FLASH acquisition from T1 and proton density maps.

Examples

```
>>> from nipyne.interfaces.freesurfer import SynthesizeFLASH
>>> syn = SynthesizeFLASH(tr=20, te=3, flip_angle=30)
>>> syn.inputs.t1_image = 'T1.mgz'
>>> syn.inputs.pd_image = 'PD.mgz'
>>> syn.inputs.out_file = 'flash_30syn.mgz'
>>> syn.cmdline
'mri_synthesize 20.00 30.00 3.000 T1.mgz PD.mgz flash_30syn.mgz'
```

Inputs:

```
[Mandatory]
flip_angle: (a float)
    flip angle (in degrees)
    flag: %.2f, position: 3
pd_image: (an existing file name)
    image of proton density values
    flag: %s, position: 6
```

```

t1_image: (an existing file name)
    image of T1 values
    flag: %s, position: 5
te: (a float)
    echo time (in msec)
    flag: %.3f, position: 4
tr: (a float)
    repetition time (in msec)
    flag: %.2f, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixed_weighting: (a boolean)
    use a fixed weighting to generate optimal gray/white contrast
    flag: -w, position: 1
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    image to write
    flag: %s
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    synthesized FLASH acquisition

```

UnpackSDICOMDir[Link to code](#)Wraps command **unpacksdcm**dirUse **unpacksdcm**dir to convert dicom filesCall **unpacksdcm**dir -help from the command line to see more information on using this command.**Examples**

```

>>> from nipy.interfaces.freesurfer import UnpackSDICOMDir
>>> unpack = UnpackSDICOMDir()
>>> unpack.inputs.source_dir = '.'
>>> unpack.inputs.output_dir = '.'
>>> unpack.inputs.run_info = (5, 'mprage', 'nii', 'struct')
>>> unpack.inputs.dir_structure = 'generic'
>>> unpack.cmdline
'unpacksdcm -generic -targ . -run 5 mprage nii struct -src .'

```

Inputs:

```
[Mandatory]
config: (an existing file name)
    specify unpacking rules in file
    flag: -cfg %s
    mutually_exclusive: run_info, config, seq_config
run_info: (a tuple of the form: (an integer (int or long), a unicode
    string, a unicode string, a unicode string))
    runno subid format name : spec unpacking rules on cmdline
    flag: -run %d %s %s %s
    mutually_exclusive: run_info, config, seq_config
seq_config: (an existing file name)
    specify unpacking rules based on sequence
    flag: -seqcfg %s
    mutually_exclusive: run_info, config, seq_config
source_dir: (an existing directory name)
    directory with the DICOM files
    flag: -src %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dir_structure: ('fsfast' or 'generic')
    unpack to specified directory structures
    flag: -%s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
log_file: (an existing file name)
    explicitly set log file
    flag: -log %s
no_info_dump: (a boolean)
    do not create infodump file
    flag: -noinfodump
no_unpack_err: (a boolean)
    do not try to unpack runs with errors
    flag: -no-unpackerr
output_dir: (a directory name)
    top directory into which the files will be unpacked
    flag: -targ %s
scan_only: (an existing file name)
    only scan the directory and put result in file
    flag: -scanonly %s
spm_zeropad: (an integer (int or long))
    set frame number zero padding width for SPM
    flag: -nspmzeropad %d
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```


Outputs:

None

WatershedSkullStrip

[Link to code](#)

Wraps command **mri_watershed**

This program strips skull and other outer non-brain tissue and produces the brain volume from T1 volume or the scanned volume.

The “watershed” segmentation algorithm was used to determine the intensity values for white matter, grey matter, and CSF. A force field was then used to fit a spherical surface to the brain. The shape of the surface fit was then evaluated against a previously derived template.

The default parameters are: -w 0.82 -b 0.32 -h 10 -seedpt -ta -wta

(Segonne 2004)

Examples

```
>>> from nipy.interfaces.freesurfer import WatershedSkullStrip
>>> skullstrip = WatershedSkullStrip()
>>> skullstrip.inputs.in_file = "T1.mgz"
>>> skullstrip.inputs.t1 = True
>>> skullstrip.inputs.transform = "transforms/talairach_with_skull.lta"
>>> skullstrip.inputs.out_file = "brainmask.auto.mgz"
>>> skullstrip.cmdline
'mri_watershed -T1 transforms/talairach_with_skull.lta T1.mgz brainmask.auto.mgz'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input volume
        flag: %s, position: -2
out_file: (a file name, nipy default value: brainmask.auto.mgz)
        output volume
        flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
brain_atlas: (an existing file name)
            flag: -brain_atlas %s, position: -4
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
subjects_dir: (an existing directory name)
              subjects directory
t1: (a boolean)
    specify T1 input volume (T1 grey value = 110)
    flag: -T1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
```

```
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
transform: (a file name)
undocumented
flag: %s, position: -3
```

Outputs:

```
out_file: (a file name)
skull stripped brain volume
```

interfaces.freesurfer.registration

EMRegister

[Link to code](#)

Wraps command `mri_em_register`

This program creates a transform in lta format

Examples

```
>>> from nipy.interfaces.freesurfer import EMRegister
>>> register = EMRegister()
>>> register.inputs.in_file = 'norm.mgz'
>>> register.inputs.template = 'aseg.mgz'
>>> register.inputs.out_file = 'norm_transform.lta'
>>> register.inputs.skull = True
>>> register.inputs.nbrspacing = 9
>>> register.cmdline
'mri_em_register -uns 9 -skull norm.mgz aseg.mgz norm_transform.lta'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        in brain volume
        flag: %s, position: -3
template: (an existing file name)
        template gca
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask: (an existing file name)
      use volume as a mask
      flag: -mask %s
nbrspacing: (an integer (int or long))
```

```

        align to atlas containing skull setting unknown_nbr_spacing =
        nbrspacing
        flag: -uns %d
num_threads: (an integer (int or long))
        allows for specifying more threads
out_file: (a file name)
        output transform
        flag: %s, position: -1
skull: (a boolean)
        align to atlas containing skull (uns=5)
        flag: -skull
subjects_dir: (an existing directory name)
        subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
transform: (an existing file name)
        Previously computed transform
        flag: -t %s

```

Outputs:

```

out_file: (a file name)
        output transform

```

MPRtoMNI305[Link to code](#)Wraps command **mpr2mni305**

For complete details, see FreeSurfer documentation

Examples

```

>>> from nipy.interfaces.freesurfer import MPRtoMNI305, Info
>>> mprtomni305 = MPRtoMNI305()
>>> mprtomni305.inputs.target = 'structural.nii'
>>> mprtomni305.inputs.reference_dir = '.'
>>> mprtomni305.cmdline
'mpr2mni305 output'
>>> mprtomni305.inputs.out_file = 'struct_out'
>>> mprtomni305.cmdline
'mpr2mni305 struct_out'
>>> mprtomni305.inputs.environ['REFDIR'] == os.path.join(Info.home(), 'average')
True
>>> mprtomni305.inputs.environ['MPR2MNI305_TARGET']
'structural'
>>> mprtomni305.run()

```

Inputs:

```

[Mandatory]
reference_dir: (an existing directory name, nipy default value: )
        TODO
target: (a string, nipy default value: )
        input atlas file

```

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (a file name, nipyne default value: )
    the input file prefix for MPRTtoMNI305
    flag: %s
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
log_file: (an existing file name, nipyne default value:
    stdout.nipyne)
    The output log
out_file: (a file name)
    The output file '<in_file>_to_<target>_t4_vox2vox.txt'
```

Paint

[Link to code](#)

Wraps command `mriscp_paint`

This program is useful for extracting one of the arrays (“a variable”) from a surface-registration template file. The output is a file containing a surface-worth of per-vertex values, saved in “curvature” format. Because the template data is sampled to a particular surface mesh, this conjures the idea of “painting to a surface”.

Examples

```
>>> from nipyne.interfaces.freesurfer import Paint
>>> paint = Paint()
>>> paint.inputs.in_surf = 'lh.pial'
>>> paint.inputs.template = 'aseg.mgz'
>>> paint.inputs.averages = 5
>>> paint.inputs.out_file = 'lh.avg_curv'
>>> paint.cmdline
'mriscp_paint -a 5 aseg.mgz lh.pial lh.avg_curv'
```

Inputs:

```
[Mandatory]
in_surf: (an existing file name)
    Surface file with grid (vertices) onto which the template data is to
    be sampled or 'painted'
    flag: %s, position: -2
template: (an existing file name)
    Template file
```

```

        flag: %s, position: -3

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
averages: (an integer (int or long))
    Average curvature patterns
    flag: -a %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    File containing a surface-worth of per-vertex values, saved in
    'curvature' format.
    flag: %s, position: -1
subjects_dir: (an existing directory name)
    subjects directory
template_param: (an integer (int or long))
    Frame number of the input template
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
    File containing a surface-worth of per-vertex values, saved in
    'curvature' format.

```

Register[Link to code](#)Wraps command **mrisc_register**

This program registers a surface to an average surface template.

Examples

```

>>> from nipy.interfaces.freesurfer import Register
>>> register = Register()
>>> register.inputs.in_surf = 'lh.pial'
>>> register.inputs.in_smoothwm = 'lh.pial'
>>> register.inputs.in_sulc = 'lh.pial'
>>> register.inputs.target = 'aseg.mgz'
>>> register.inputs.out_file = 'lh.pial.reg'
>>> register.inputs.curv = True
>>> register.cmdline
'mrisc_register -curv lh.pial aseg.mgz lh.pial.reg'

```

Inputs:

```

[Mandatory]
in_sulc: (an existing file name)
    Undocumented mandatory input file
    ${SUBJECTS_DIR}/surf/{hemisphere}.sulc
in_surf: (an existing file name)
    Surface to register, often {hemi}.sphere
    flag: %s, position: -3
target: (an existing file name)
    The data to register to. In normal recon-all usage, this is a
    template file for average surface.
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
curv: (a boolean)
    Use smoothwm curvature for final alignment
    flag: -curv
    requires: in_smoothwm
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_smoothwm: (an existing file name)
    Undocumented input file ${SUBJECTS_DIR}/surf/{hemisphere}.smoothwm
out_file: (a file name)
    Output surface file to capture registration
    flag: %s, position: -1
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
    Output surface file to capture registration

```

RegisterAVItoTalairach[Link to code](#)Wraps command **avi2talxfm**

converts the vox2vox from talairach_avi to a talairach.xfm file

This is a script that converts the vox2vox from talairach_avi to a talairach.xfm file. It is meant to replace the following cmd line:

```

tkregister2_cmdl -mov $InVol -targ $FREESURFER_HOME/average/mni305.cor.mgz -
xfmout ${XFM} -vox2vox talsrcimg_to_${target}_t4_vox2vox.txt -noedit -reg tal-
srcimg.reg.tmp.dat set targ = $FREESURFER_HOME/average/mni305.cor.mgz set subject
= mgh-02407836-v2 set InVol = $SUBJECTS_DIR/$subject/mri/orig.mgz set vox2vox =
$SUBJECTS_DIR/$subject/mri/transforms/talsrcimg_to_711-2C_as_mni_average_305_t4_vox2vox.txt

```

Examples

```
>>> from nipy.interfaces.freesurfer import RegisterAVItaTalairach
>>> register = RegisterAVItaTalairach()
>>> register.inputs.in_file = 'structural.mgz'
>>> register.inputs.target = 'mni305.cor.mgz'
>>> register.inputs.vox2vox = 'talsrcimg_to_structural_t4_vox2vox.txt'
>>> register.cmdline
'avi2talxfm structural.mgz mni305.cor.mgz talsrcimg_to_structural_t4_vox2vox.txt_
↳talairach.auto.xfm'
```

```
>>> register.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    The input file
    flag: %s, position: 0
target: (an existing file name)
    The target file
    flag: %s, position: 1
vox2vox: (an existing file name)
    The vox2vox file
    flag: %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name, nipy default value: talairach.auto.xfm)
    The transform output
    flag: %s, position: 3
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
log_file: (an existing file name, nipy default value:
    stdout.nipy)
    The output log
out_file: (a file name)
    The output file for RegisterAVItaTalairach
```

interfaces.freesurfer.utils

AddXFormToHeader

[Link to code](#)

Wraps command `mri_add_xform_to_header`

Just adds specified xform to the volume header

(!) **WARNING:** transform input **MUST** be an absolute path to a DataSink'ed transform or the output will reference a transform in the workflow cache directory!

```
>>> from nipy.interfaces.freesurfer import AddXFormToHeader
>>> adder = AddXFormToHeader()
>>> adder.inputs.in_file = 'norm.mgz'
>>> adder.inputs.transform = 'trans.mat'
>>> adder.cmdline
'mri_add_xform_to_header trans.mat norm.mgz output.mgz'
```

```
>>> adder.inputs.copy_name = True
>>> adder.cmdline
'mri_add_xform_to_header -c trans.mat norm.mgz output.mgz'
```

```
>>> adder.run()
```

References:

[https://surfer.nmr.mgh.harvard.edu/fswiki/mri_add_xform_to_header]

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input volume
        flag: %s, position: -2
transform: (a file name)
          xfm file
          flag: %s, position: -3

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
copy_name: (a boolean)
          do not try to load the xfmfile, just copy name
          flag: -c
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
out_file: (a file name, nipy default value: output.mgz)
          output volume
          flag: %s, position: -1
subjects_dir: (an existing directory name)
              subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
```



```
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    be verbose
flag: -v
```

Outputs:

```
out_file: (an existing file name)
    output volume
```

Aparc2Aseg[Link to code](#)**Wraps command `mri_aparc2aseg`**

Maps the cortical labels from the automatic cortical parcellation (aparc) to the automatic segmentation volume (aseg). The result can be used as the aseg would. The algorithm is to find each aseg voxel labeled as cortex (3 and 42) and assign it the label of the closest cortical vertex. If the voxel is not in the ribbon (as defined by mri/ lh.ribbon and rh.ribbon), then the voxel is marked as unknown (0). This can be turned off with `-noribbon`. The cortical parcellation is obtained from subject/label/hemi.aparc.annot which should be based on the curvature.buckner40.filled.desikan_killiany.gcs atlas. The aseg is obtained from subject/mri/aseg.mgz and should be based on the RB40_talairach_2005-07-20.gca atlas. If these atlases are used, then the segmentations can be viewed with tkmedit and the FreeSurferColorLUT.txt color table found in \$FREESURFER_HOME. These are the default atlases used by recon-all.

Examples

```
>>> from nipy.interfaces.freesurfer import Aparc2Aseg
>>> aparc2aseg = Aparc2Aseg()
>>> aparc2aseg.inputs.lh_white = 'lh.pial'
>>> aparc2aseg.inputs.rh_white = 'lh.pial'
>>> aparc2aseg.inputs.lh_pial = 'lh.pial'
>>> aparc2aseg.inputs.rh_pial = 'lh.pial'
>>> aparc2aseg.inputs.lh_ribbon = 'label.mgz'
>>> aparc2aseg.inputs.rh_ribbon = 'label.mgz'
>>> aparc2aseg.inputs.ribbon = 'label.mgz'
>>> aparc2aseg.inputs.lh_annotation = 'lh.pial'
>>> aparc2aseg.inputs.rh_annotation = 'lh.pial'
>>> aparc2aseg.inputs.out_file = 'aparc+aseg.mgz'
>>> aparc2aseg.inputs.label_wm = True
>>> aparc2aseg.inputs.rip_unknown = True
>>> aparc2aseg.cmdline
'mri_aparc2aseg --labelwm --o aparc+aseg.mgz --rip-unknown --s subject_id'
```

Inputs:

```
[Mandatory]
lh_annotation: (an existing file name)
    Input file must be <subject_id>/label/lh.aparc.annot
lh_pial: (an existing file name)
    Input file must be <subject_id>/surf/lh.pial
lh_ribbon: (an existing file name)
    Input file must be <subject_id>/mri/lh.ribbon.mgz
lh_white: (an existing file name)
    Input file must be <subject_id>/surf/lh.white
out_file: (a file name)
    Full path of file to save the output segmentation in
```

```
    flag: --o %s
rh_annotation: (an existing file name)
    Input file must be <subject_id>/label/rh.aparc.annot
rh_pial: (an existing file name)
    Input file must be <subject_id>/surf/rh.pial
rh_ribbon: (an existing file name)
    Input file must be <subject_id>/mri/rh.ribbon.mgz
rh_white: (an existing file name)
    Input file must be <subject_id>/surf/rh.white
ribbon: (an existing file name)
    Input file must be <subject_id>/mri/ribbon.mgz
subject_id: (a string, nipy default value: subject_id)
    Subject being processed
    flag: --s %s

[Optional]
a2009s: (a boolean)
    Using the a2009s atlas
    flag: --a2009s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
aseg: (an existing file name)
    Input aseg file
    flag: --aseg %s
copy_inputs: (a boolean)
    If running as a node, set this to True.This will copy the input
    files to the node directory.
ctxseg: (an existing file name)
    flag: --ctxseg %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
filled: (an existing file name)
    Implicit input filled file. Only required with FS v5.3.
hypo_wm: (a boolean)
    Label hypointensities as WM
    flag: --hypo-as-wm
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
label_wm: (a boolean)
    For each voxel labeled as white matter in the aseg, re-assign
    its label to be that of the closest cortical point if its
    distance is less than dmaxctx
    flag: --labelwm
rip_unknown: (a boolean)
    Do not label WM based on 'unknown' corical label
    flag: --rip-unknown
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
volmask: (a boolean)
    Volume mask flag
```

```
flag: --volmask
```

Outputs:

```
out_file: (a file name)
          Output aseg file
flag: %s
```

Apas2Aseg[Link to code](#)Wraps command **apas2aseg**

Converts aparc+aseg.mgz into something like aseg.mgz by replacing the cortical segmentations 1000-1035 with 3 and 2000-2035 with 42. The advantage of this output is that the cortical label conforms to the actual surface (this is not the case with aseg.mgz).

Examples

```
>>> from nipy.interfaces.freesurfer import Apas2Aseg
>>> apas2aseg = Apas2Aseg()
>>> apas2aseg.inputs.in_file = 'aseg.mgz'
>>> apas2aseg.inputs.out_file = 'output.mgz'
>>> apas2aseg.cmdline
'apas2aseg --i aseg.mgz --o output.mgz'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
          Input aparc+aseg.mgz
          flag: --i %s
out_file: (a file name)
          Output aseg file
          flag: --o %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
subjects_dir: (an existing directory name)
               subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          Output aseg file
flag: %s
```

ApplyMask

[Link to code](#)

Wraps command **mri_mask**

Use Freesurfer's **mri_mask** to apply a mask to an image.

The mask file need not be binarized; it can be thresholded above a given value before application. It can also optionally be transformed into input space with an LTA matrix.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input image (will be masked)
        flag: %s, position: -3
mask_file: (an existing file name)
        image defining mask space
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
invert_xfm: (a boolean)
            invert transformation
            flag: -invert
keep_mask_deletion_edits: (a boolean)
                           transfer voxel-deletion edits (voxels=1) from mask to out vol
                           flag: -keep_mask_deletion_edits
mask_thresh: (a float)
              threshold mask before applying
              flag: -T %.4f
out_file: (a file name)
          final image to write
          flag: %s, position: -1
subjects_dir: (an existing directory name)
              subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
transfer: (an integer (int or long))
          transfer only voxel value # from mask to out
          flag: -transfer %d
use_abs: (a boolean)
         take absolute value of mask before applying
         flag: -abs
xfm_file: (an existing file name)
          LTA-format transformation matrix to align mask with input
          flag: -xform %s
xfm_source: (an existing file name)
            image defining transform source space
            flag: -lta_src %s
xfm_target: (an existing file name)
```

```
image defining transform target space
flag: -lta_dst %s
```

Outputs:

```
out_file: (an existing file name)
masked image
```

CheckTalairachAlignment

[Link to code](#)

Wraps command **talairach_afd**

This program detects Talairach alignment failures

Examples

```
>>> from nipy.interfaces.freesurfer import CheckTalairachAlignment
>>> checker = CheckTalairachAlignment()
```

```
>>> checker.inputs.in_file = 'trans.mat'
>>> checker.inputs.threshold = 0.005
>>> checker.cmdline
'talairach_afd -T 0.005 -xfm trans.mat'
```

```
>>> checker.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    specify the talairach.xfm file to check
    flag: -xfm %s, position: -1
    mutually_exclusive: subject
subject: (a string)
    specify subject's name
    flag: -subj %s, position: -1
    mutually_exclusive: in_file

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (a float)
```

```
Talairach transforms for subjects with p-values <= T are considered
as very unlikely default=0.010
flag: -T %.3f
```

Outputs:

```
out_file: (a file name)
    The input file for CheckTalairachAlignment
```

Contrast[Link to code](#)Wraps command **pctsurfcon**

Compute surface-wise gray/white contrast

Examples

```
>>> from nipy.interfaces.freesurfer import Contrast
>>> contrast = Contrast()
>>> contrast.inputs.subject_id = '10335'
>>> contrast.inputs.hemisphere = 'lh'
>>> contrast.inputs.white = 'lh.white'
>>> contrast.inputs.thickness = 'lh.thickness'
>>> contrast.inputs.annotation = '../label/lh.aparc.annot'
>>> contrast.inputs.cortex = '../label/lh.cortex.label'
>>> contrast.inputs.rawavg = '../mri/rawavg.mgz'
>>> contrast.inputs.orig = '../mri/orig.mgz'
>>> contrast.cmdline
'pctsurfcon --lh-only --s 10335'
```

Inputs:

```
[Mandatory]
annotation: (a file name)
    Input annotation file must be
    <subject_id>/label/<hemisphere>.aparc.annot
cortex: (a file name)
    Input cortex label must be
    <subject_id>/label/<hemisphere>.cortex.label
hemisphere: ('lh' or 'rh')
    Hemisphere being processed
    flag: --%s-only
orig: (an existing file name)
    Implicit input file mri/orig.mgz
rawavg: (an existing file name)
    Implicit input file mri/rawavg.mgz
subject_id: (a string, nipy default value: subject_id)
    Subject being processed
    flag: --s %s
thickness: (an existing file name)
    Input file must be <subject_id>/surf/?h.thickness
white: (an existing file name)
    Input file must be <subject_id>/surf/<hemisphere>.white

[Optional]
args: (a unicode string)
    Additional parameters to the command
```

```

    flag: %s
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the input
    files to the node directory.
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_contrast: (a file name)
    Output contrast file from Contrast
out_log: (an existing file name)
    Output log from Contrast
out_stats: (a file name)
    Output stats file from Contrast

```

Curvature[Link to code](#)Wraps command **mr_{is}_curvature**

This program will compute the second fundamental form of a cortical surface. It will create two new files <hemi>.<surface>.H and <hemi>.<surface>.K with the mean and Gaussian curvature respectively.

Examples

```

>>> from nipy.interfaces.freesurfer import Curvature
>>> curv = Curvature()
>>> curv.inputs.in_file = 'lh.pial'
>>> curv.inputs.save = True
>>> curv.cmdline
'mris_curvature -w lh.pial'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    Input file for Curvature
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
averages: (an integer (int or long))
    Perform this number iterative averages of curvature measure before
    saving

```

```
    flag: -a %d
copy_input: (a boolean)
    Copy input file to current directory
distances: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    Undocumented input integer distances
    flag: -distances %d %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
n: (a boolean)
    Undocumented boolean flag
    flag: -n
save: (a boolean)
    Save curvature files (will only generate screen output without this
    option)
    flag: -w
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (a float)
    Undocumented input threshold
    flag: -thresh %.3f
```

Outputs:

```
out_gauss: (a file name)
    Gaussian curvature output file
out_mean: (a file name)
    Mean curvature output file
```

CurvatureStats

[Link to code](#)

Wraps command **mriscurvature_stats**

In its simplest usage, 'mriscurvature_stats' will compute a set of statistics on its input <curvFile>. These statistics are the mean and standard deviation of the particular curvature on the surface, as well as the results from several surface-based integrals.

Additionally, 'mriscurvature_stats' can report the max/min curvature values, and compute a simple histogram based on all curvature values.

Curvatures can also be normalised and constrained to a given range before computation.

Principal curvature (K, H, k1 and k2) calculations on a surface structure can also be performed, as well as several functions derived from k1 and k2.

Finally, all output to the console, as well as any new curvatures that result from the above calculations can be saved to a series of text and binary-curvature files.

Examples

```
>>> from nipy.interfaces.freesurfer import CurvatureStats
>>> curvstats = CurvatureStats()
>>> curvstats.inputs.hemisphere = 'lh'
>>> curvstats.inputs.curvfile1 = 'lh.pial'
>>> curvstats.inputs.curvfile2 = 'lh.pial'
>>> curvstats.inputs.surface = 'lh.pial'
>>> curvstats.inputs.out_file = 'lh.curv.stats'
>>> curvstats.inputs.values = True
>>> curvstats.inputs.min_max = True
>>> curvstats.inputs.write = True
>>> curvstats.cmdline
'mris_curvature_stats -m -o lh.curv.stats -F pial -G --writeCurvatureFiles_
↳subject_id lh pial pial'
```

Inputs:

```
[Mandatory]
curvfile1: (an existing file name)
    Input file for CurvatureStats
    flag: %s, position: -2
curvfile2: (an existing file name)
    Input file for CurvatureStats
    flag: %s, position: -1
hemisphere: ('lh' or 'rh')
    Hemisphere being processed
    flag: %s, position: -3
subject_id: (a string, nipy default value: subject_id)
    Subject being processed
    flag: %s, position: -4

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the input
    files to the node directory.
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
min_max: (a boolean)
    Output min / max information for the processed curvature.
    flag: -m
out_file: (a file name)
    Output curvature stats file
    flag: -o %s
subjects_dir: (an existing directory name)
    subjects directory
surface: (an existing file name)
    Specify surface file for CurvatureStats
    flag: -F %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
```

```

        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
values: (a boolean)
        Triggers a series of derived curvature values
flag: -G
write: (a boolean)
        Write curvature files
flag: --writeCurvatureFiles

```

Outputs:

```

out_file: (a file name)
        Output curvature stats file

```

EulerNumber

[Link to code](#)Wraps command **mr**is_euler_number

This program computes EulerNumber for a cortical surface

Examples

```

>>> from nipy.interfaces.freesurfer import EulerNumber
>>> ft = EulerNumber()
>>> ft.inputs.in_file = 'lh.pial'
>>> ft.cmdline
'mris_euler_number lh.pial'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        Input file for EulerNumber
        flag: %s, position: -1

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
subjects_dir: (an existing directory name)
        subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
        Output file for EulerNumber

```

ExtractMainComponent

[Link to code](#)

Wraps command **mr**is_extract_main_component

Extract the main component of a tessellated surface

Examples

```
>>> from nipy.interfaces.freesurfer import ExtractMainComponent
>>> mcmp = ExtractMainComponent(in_file='lh.pial')
>>> mcmp.cmdline
'mris_extract_main_component lh.pial lh.maincmp'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input surface file
        flag: %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          surface containing main component
          flag: %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          surface containing main component
```

FixTopology

[Link to code](#)

Wraps command **mr**is_fix_topology

This program computes a mapping from the unit sphere onto the surface of the cortex from a previously generated approximation of the cortical surface, thus guaranteeing a topologically correct surface.

Examples

```
>>> from nipy.interfaces.freesurfer import FixTopology
>>> ft = FixTopology()
>>> ft.inputs.in_orig = 'lh.orig'
>>> ft.inputs.in_inflated = 'lh.inflated'
```

```
>>> ft.inputs.sphere = 'lh.qsphere.nofix'
>>> ft.inputs.hemisphere = 'lh'
>>> ft.inputs.subject_id = '10335'
>>> ft.inputs.mgz = True
>>> ft.inputs.ga = True
>>> ft.cmdline
'mris_fix_topology -ga -mgz -sphere qsphere.nofix 10335 lh'
```

Inputs:

```
[Mandatory]
copy_inputs: (a boolean)
    If running as a node, set this to True otherwise, the topology
    fixing will be done in place.
hemisphere: (a string)
    Hemisphere being processed
    flag: %s, position: -1
in_brain: (an existing file name)
    Implicit input brain.mgz
in_inflated: (an existing file name)
    Undocumented input file <hemisphere>.inflated
in_orig: (an existing file name)
    Undocumented input file <hemisphere>.orig
in_wm: (an existing file name)
    Implicit input wm.mgz
subject_id: (a string, nipyne default value: subject_id)
    Subject being processed
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ga: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    flag: -ga
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mgz: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    flag: -mgz
seed: (an integer (int or long))
    Seed for setting random number generator
    flag: -seed %d
sphere: (a file name)
    Sphere input file
    flag: -sphere %s
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
```

```
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          Output file for FixTopology
```

Jacobian[Link to code](#)Wraps command **mrjs_jacobian**

This program computes the Jacobian of a surface mapping.

Examples

```
>>> from nipy.interfaces.freesurfer import Jacobian
>>> jacobian = Jacobian()
>>> jacobian.inputs.in_origsurf = 'lh.pial'
>>> jacobian.inputs.in_mappedsurf = 'lh.pial'
>>> jacobian.cmdline
'mrjs_jacobian lh.pial lh.pial lh.jacobian'
```

Inputs:

```
[Mandatory]
in_mappedsurf: (an existing file name)
               Mapped surface
               flag: %s, position: -2
in_origsurf: (an existing file name)
              Original surface
              flag: %s, position: -3

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
           Output Jacobian of the surface mapping
           flag: %s, position: -1
subjects_dir: (an existing directory name)
              subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          Output Jacobian of the surface mapping
```

MRIFill

[Link to code](#)

Wraps command **mri_fill**

This program creates hemispheric cutting planes and fills white matter with specific values for subsequent surface tessellation.

Examples

```
>>> from nipy.interfaces.freesurfer import MRIFill
>>> fill = MRIFill()
>>> fill.inputs.in_file = 'wm.mgz'
>>> fill.inputs.out_file = 'filled.mgz'
>>> fill.cmdline
'mri_fill wm.mgz filled.mgz'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Input white matter file
        flag: %s, position: -2
out_file: (a file name)
        Output filled volume file name for MRIFill
        flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
log_file: (a file name)
          Output log file for MRIFill
          flag: -a %s
segmentation: (an existing file name)
              Input segmentation file for MRIFill
              flag: -segmentation %s
subjects_dir: (an existing directory name)
              subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
transform: (an existing file name)
           Input transform file for MRIFill
           flag: -xform %s
```

Outputs:

```
log_file: (a file name)
          Output log file from MRIFill
out_file: (a file name)
          Output file from MRIFill
```

MRIMarchingCubes

[Link to code](#)

Wraps command `mri_mc`

Uses Freesurfer's `mri_mc` to create surfaces by tessellating a given input volume

Example

```
>>> import nipy.interfaces.freesurfer as fs
>>> mc = fs.MRIMarchingCubes()
>>> mc.inputs.in_file = 'aseg.mgz'
>>> mc.inputs.label_value = 17
>>> mc.inputs.out_file = 'lh.hippocampus'
>>> mc.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    Input volume to tessellate voxels from.
    flag: %s, position: 1
label_value: (an integer (int or long))
    Label value which to tessellate from the input volume. (integer, if
    input is "filled.mgz" volume, 127 is rh, 255 is lh)
    flag: %d, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
connectivity_value: (an integer (int or long), nipy default value:
    1)
    Alter the marching cubes connectivity: 1=6+,2=18,3=6,4=26
    (default=1)
    flag: %d, position: -1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    output filename or True to generate one
    flag: ./%s, position: -2
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
surface: (an existing file name)
    binary surface of the tessellation
```

MRIPretess

[Link to code](#)

Wraps command **mri_pre**tes

Uses Freesurfer's mri_pre

Description

Changes white matter (WM) segmentation so that the neighbors of all voxels labeled as WM have a face in common - no edges or corners allowed.

Example

```
>>> import nipy.interfaces.freesurfer as fs
>>> pretess = fs.MRIPretess()
>>> pretess.inputs.in_filled = 'wm.mgz'
>>> pretess.inputs.in_norm = 'norm.mgz'
>>> pretess.inputs.nocorners = True
>>> pretess.cmdline
'mri_pre -nocorners wm.mgz wm norm.mgz wm_pre_tesswm.mgz'
>>> pretess.run()
```

Inputs:

```
[Mandatory]
in_filled: (an existing file name)
            filled volume, usually wm.mgz
            flag: %s, position: -4
in_norm: (an existing file name)
            the normalized, brain-extracted T1w image. Usually norm.mgz
            flag: %s, position: -2
label: (a unicode string or an integer (int or long), nipy default
        value: wm)
        label to be picked up, can be a Freesurfer's string like 'wm' or a
        label value (e.g. 127 for rh or 255 for lh)
        flag: %s, position: -3

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
keep: (a boolean)
        keep WM edits
        flag: -keep
nocorners: (a boolean)
            do not remove corner configurations in addition to edge ones.
            flag: -nocorners
out_file: (a file name)
            the output file after mri_pre_tess.
            flag: %s, position: -1
subjects_dir: (an existing directory name)
```



```

    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
test: (a boolean)
    adds a voxel that should be removed by mri_pretest. The value of the
    voxel is set to that of an ON-edited WM, so it should be kept with
    -keep. The output will NOT be saved.
flag: -test

```

Outputs:

```

out_file: (an existing file name)
    output file after mri_pretest

```

MRI Tessellate[Link to code](#)Wraps command **mri_tessellate**Uses Freesurfer's **mri_tessellate** to create surfaces by tessellating a given input volume**Example**

```

>>> import nipy.interfaces.freesurfer as fs
>>> tess = fs.MRI Tessellate()
>>> tess.inputs.in_file = 'aseg.mgz'
>>> tess.inputs.label_value = 17
>>> tess.inputs.out_file = 'lh.hippocampus'
>>> tess.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    Input volume to tessellate voxels from.
    flag: %s, position: -3
label_value: (an integer (int or long))
    Label value which to tessellate from the input volume. (integer, if
    input is "filled.mgz" volume, 127 is rh, 255 is lh)
    flag: %d, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    output filename or True to generate one
    flag: %s, position: -1
subjects_dir: (an existing directory name)

```

```

    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tessellate_all_voxels: (a boolean)
    Tessellate the surface of all voxels with different labels
    flag: -a
use_real_RAS_coordinates: (a boolean)
    Saves surface with real RAS coordinates where c_(r,a,s) != 0
    flag: -n

```

Outputs:

```

surface: (an existing file name)
    binary surface of the tessellation

```

MRIsCalc[Link to code](#)Wraps command **mriscalc**

'mriscalc' is a simple calculator that operates on FreeSurfer curvatures and volumes. In most cases, the calculator functions with three arguments: two inputs and an <ACTION> linking them. Some actions, however, operate with only one input <file1>. In all cases, the first input <file1> is the name of a FreeSurfer curvature overlay (e.g. rh.curv) or volume file (e.g. orig.mgz). For two inputs, the calculator first assumes that the second input is a file. If, however, this second input file doesn't exist, the calculator assumes it refers to a float number, which is then processed according to <ACTION>. Note: <file1> and <file2> should typically be generated on the same subject.

Examples

```

>>> from nipy.interfaces.freesurfer import MRIsCalc
>>> example = MRIsCalc()
>>> example.inputs.in_file1 = 'lh.area'
>>> example.inputs.in_file2 = 'lh.area.pial'
>>> example.inputs.action = 'add'
>>> example.inputs.out_file = 'area.mid'
>>> example.cmdline
'mriscalc -o lh.area.mid lh.area add lh.area.pial'

```

Inputs:

```

[Mandatory]
action: (a string)
    Action to perform on input file(s)
    flag: %s, position: -2
in_file1: (an existing file name)
    Input file 1
    flag: %s, position: -3
out_file: (a file name)
    Output file after calculation
    flag: -o %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s

```

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
         Print an error message instead of throwing an exception in case the
         interface fails to run
in_file2: (an existing file name)
         Input file 2
         flag: %s, position: -1
         mutually_exclusive: in_float, in_int
in_float: (a float)
         Input float
         flag: %f, position: -1
         mutually_exclusive: in_file2, in_int
in_int: (an integer (int or long))
         Input integer
         flag: %d, position: -1
         mutually_exclusive: in_file2, in_float
subjects_dir: (an existing directory name)
         subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
         Control terminal output: `stream` - displays to terminal immediately
         (default), `allatonce` - waits till command is finished to display
         output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
         Output file after calculation

```

MRIsCombine[Link to code](#)**Wraps command `mrisc`**Uses `FreeSurfer's mrisc` to combine two surface files into one.For complete details, see the [mrisc Documentation](#).

If given an `out_file` that does not begin with 'lh.' or 'rh.', `mrisc` will prepend 'lh.' to the file name. To avoid this behavior, consider setting `out_file = './<filename>'`, or leaving `out_file` blank.

In a Node/Workflow, `out_file` is interpreted literally.**Example**

```

>>> import nipy.interfaces.freesurfer as fs
>>> mris = fs.MRIsCombine()
>>> mris.inputs.in_files = ['lh.pial', 'rh.pial']
>>> mris.inputs.out_file = 'bh.pial'
>>> mris.cmdline
'mrisc --combinesurfs lh.pial rh.pial bh.pial'
>>> mris.run()

```

Inputs:

```

[Mandatory]
in_files: (a list of from 2 to 2 items which are a file name)
         Two surfaces to be combined.

```

```
    flag: --combinesurfs %s, position: 1
out_file: (a file name)
    Output filename. Combined surfaces from in_files.
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    Output filename. Combined surfaces from in_files.
```

MRIsConvert

[Link to code](#)

Wraps command **mrisc**onvert

Uses Freesurfer's **mrisc**onvert to convert surface files to various formats

Example

```
>>> import nipy.interfaces.freesurfer as fs
>>> mris = fs.MRIsConvert()
>>> mris.inputs.in_file = 'lh.pial'
>>> mris.inputs.out_datatype = 'gii'
>>> mris.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    File to read/convert
    flag: %s, position: -2
out_datatype: ('asc' or 'ico' or 'tri' or 'stl' or 'vtk' or 'gii' or
    'mgh' or 'mgz')
    These file formats are supported: ASCII: .asc ICO: .ico, .tri GEO:
    .geo STL: .stl VTK: .vtk GIFTI: .gii MGH surface-encoded 'volume':
    .mgh, .mgz
    mutually_exclusive: out_file
out_file: (a file name)
    output filename or True to generate one
    flag: %s, position: -1
```

```

        mutually_exclusive: out_datatype

[Optional]
annot_file: (an existing file name)
    input is annotation or gifti label data
    flag: --annot %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dataarray_num: (an integer (int or long))
    if input is gifti, 'num' specifies which data array to use
    flag: --da_num %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
functional_file: (an existing file name)
    input is functional time-series or other multi-frame data (must
    specify surface)
    flag: -f %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
label_file: (an existing file name)
    infile is .label file, label is name of this label
    flag: --label %s
labelstats_outfile: (a file name)
    outfile is name of gifti file to which label stats will be written
    flag: --labelstats %s
normal: (a boolean)
    output is an ascii file where vertex data
    flag: -n
origname: (a string)
    read orig positions
    flag: -o %s
parcstats_file: (an existing file name)
    infile is name of text file containing label/val pairs
    flag: --parcstats %s
patch: (a boolean)
    input is a patch, not a full surface
    flag: -p
rescale: (a boolean)
    rescale vertex xyz so total area is same as group average
    flag: -r
scalarcurv_file: (an existing file name)
    input is scalar curv overlay file (must still specify surface)
    flag: -c %s
scale: (a float)
    scale vertex xyz by scale
    flag: -s %.3f
subjects_dir: (an existing directory name)
    subjects directory
talairachxfm_subjid: (a string)
    apply talairach xfm of subject to vertex xyz
    flag: -t %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display

```

```

        output, `file` - writes output to file, `none` - output is ignored
to_scanner: (a boolean)
    convert coordinates from native FS (tkr) coords to scanner coords
    flag: --to-scanner
to_tkr: (a boolean)
    convert coordinates from scanner coords to native FS (tkr) coords
    flag: --to-tkr
vertex: (a boolean)
    Writes out neighbors of a vertex in each row
    flag: -v
xyz_ascii: (a boolean)
    Print only surface xyz to ascii file
    flag: -a

```

Outputs:

```

converted: (an existing file name)
    converted output surface

```

MRIsExpand[Link to code](#)Wraps command **mris_expand**

Expands a surface (typically ?h.white) outwards while maintaining smoothness and self-intersection constraints.

Examples

```

>>> from nipy.interfaces.freesurfer import MRIsExpand
>>> mris_expand = MRIsExpand(thickness=True, distance=0.5)
>>> mris_expand.inputs.in_file = 'lh.white'
>>> mris_expand.cmdline
'mris_expand -thickness lh.white 0.5 expanded'
>>> mris_expand.inputs.out_name = 'graymid'
>>> mris_expand.cmdline
'mris_expand -thickness lh.white 0.5 graymid'

```

Inputs:

```

[Mandatory]
distance: (a float)
    Distance in mm or fraction of cortical thickness
    flag: %g, position: -2
in_file: (an existing file name)
    Surface to expand
    flag: %s, position: -3

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dt: (a float)
    dt (implicit: 0.25)
    flag: -T %g
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
nsurfaces: (an integer (int or long))
    Number of surfaces to write during expansion
    flag: -N %d
out_name: (a unicode string, nipy default value: expanded)
    Output surface file
    If no path, uses directory of `in_file`
    If no path AND missing "lh." or "rh.", derive from `in_file`
    flag: %s, position: -1
pial: (a unicode string)
    Name of pial file (implicit: "pial")
    If no path, uses directory of `in_file`
    If no path AND missing "lh." or "rh.", derive from `in_file`
    flag: -pial %s
smooth_averages: (an integer (int or long))
    Smooth surface with N iterations after expansion
    flag: -A %d
sphere: (a unicode string, nipy default value: sphere)
    WARNING: Do not change this trait
spring: (a float)
    Spring term (implicit: 0.05)
    flag: -S %g
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thickness: (a boolean)
    Expand by fraction of cortical thickness, not mm
    flag: -thickness
thickness_name: (a unicode string)
    Name of thickness file (implicit: "thickness")
    If no path, uses directory of `in_file`
    If no path AND missing "lh." or "rh.", derive from `in_file`
    flag: -thickness_name %s
write_iterations: (an integer (int or long))
    Write snapshots of expansion every N iterations
    flag: -W %d

```

Outputs:

```

out_file: (a file name)
    Output surface file

```

MRIsInflate[Link to code](#)Wraps command **mriss_inflate**

This program will inflate a cortical surface.

Examples

```

>>> from nipy.interfaces.freesurfer import MRIsInflate
>>> inflate = MRIsInflate()

```

```
>>> inflate.inputs.in_file = 'lh.pial'
>>> inflate.inputs.no_save_sulc = True
>>> inflate.cmdline
'mris_inflate -no-save-sulc lh.pial lh.inflated'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Input file for MRIsInflate
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
no_save_sulc: (a boolean)
              Do not save sulc file as output
              flag: -no-save-sulc
              mutually_exclusive: out_sulc
out_file: (a file name)
          Output file for MRIsInflate
          flag: %s, position: -1
out_sulc: (a file name)
          Output sulc file
          mutually_exclusive: no_save_sulc
subjects_dir: (an existing directory name)
              subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          Output file for MRIsInflate
out_sulc: (a file name)
          Output sulc file
```

MakeAverageSubject

[Link to code](#)

Wraps command **make_average_subject**

Make an average freesurfer subject

Examples

```
>>> from nipy.interfaces.freesurfer import MakeAverageSubject
>>> avg = MakeAverageSubject(subjects_ids=['s1', 's2'])
```



```
>>> avg.cmdline
'make_average_subject --out average --subjects s1 s2'
```

Inputs:

```
[Mandatory]
subjects_ids: (a list of items which are a unicode string)
               freesurfer subjects ids to average
               flag: --subjects %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_name: (a file name, nipy default value: average)
          name for the average subject
          flag: --out %s
subjects_dir: (an existing directory name)
              subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
average_subject_name: (a unicode string)
                     Output registration file
```

MakeSurfaces

[Link to code](#)**Wraps command `mris_make_surfaces`**

This program positions the tessellation of the cortical surface at the white matter surface, then the gray matter surface and generate surface files for these surfaces as well as a ‘curvature’ file for the cortical thickness, and a surface file which approximates layer IV of the cortical sheet.

Examples

```
>>> from nipy.interfaces.freesurfer import MakeSurfaces
>>> makesurfaces = MakeSurfaces()
>>> makesurfaces.inputs.hemisphere = 'lh'
>>> makesurfaces.inputs.subject_id = '10335'
>>> makesurfaces.inputs.in_orig = 'lh.pial'
>>> makesurfaces.inputs.in_wm = 'wm.mgz'
>>> makesurfaces.inputs.in_filled = 'norm.mgz'
>>> makesurfaces.inputs.in_label = 'aparc+aseg.nii'
>>> makesurfaces.inputs.in_T1 = 'T1.mgz'
>>> makesurfaces.inputs.orig_pial = 'lh.pial'
>>> makesurfaces.cmdline
```

```
'mris_make_surfaces -T1 T1.mgz -orig pial -orig_pial pial 10335 lh'
```

Inputs:

```
[Mandatory]
hemisphere: ('lh' or 'rh')
    Hemisphere being processed
    flag: %s, position: -1
in_filled: (an existing file name)
    Implicit input file filled.mgz
in_orig: (an existing file name)
    Implicit input file <hemisphere>.orig
    flag: -orig %s
in_wm: (an existing file name)
    Implicit input file wm.mgz
subject_id: (a string, nipy default value: subject_id)
    Subject being processed
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the input
    files to the node directory.
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fix_mtl: (a boolean)
    Undocumented flag
    flag: -fix_mtl
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_T1: (an existing file name)
    Input brain or T1 file
    flag: -T1 %s
in_aseg: (an existing file name)
    Input segmentation file
    flag: -aseg %s
in_label: (an existing file name)
    Implicit input label/<hemisphere>.aparc.annot
    mutually_exclusive: noaparc
in_white: (an existing file name)
    Implicit input that is sometimes used
longitudinal: (a boolean)
    No documentation (used for longitudinal processing)
    flag: -long
maximum: (a float)
    No documentation (used for longitudinal processing)
    flag: -max %.1f
mgz: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    flag: -mgz
no_white: (a boolean)
```

```

    Undocumented flag
    flag: -nowhite
noaparc: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    flag: -noaparc
    mutually_exclusive: in_label
orig_pial: (an existing file name)
    Specify a pial surface to start with
    flag: -orig_pial %s
    requires: in_label
orig_white: (an existing file name)
    Specify a white surface to start with
    flag: -orig_white %s
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
white: (a string)
    White surface name
    flag: -white %s
white_only: (a boolean)
    Undocumented flag
    flag: -whiteonly

```

Outputs:

```

out_area: (a file name)
    Output area file for MakeSurfaces
out_cortex: (a file name)
    Output cortex file for MakeSurfaces
out_curv: (a file name)
    Output curv file for MakeSurfaces
out_pial: (a file name)
    Output pial surface for MakeSurfaces
out_thickness: (a file name)
    Output thickness file for MakeSurfaces
out_white: (a file name)
    Output white matter hemisphere surface

```

ParcellationStats[Link to code](#)Wraps command **mrisc_anatomical_stats**

This program computes a number of anatomical properties.

Examples

```

>>> from nipy.interfaces.freesurfer import ParcellationStats
>>> import os
>>> parccstats = ParcellationStats()
>>> parccstats.inputs.subject_id = '10335'
>>> parccstats.inputs.hemisphere = 'lh'
>>> parccstats.inputs.wm = '../mri/wm.mgz'
>>> parccstats.inputs.transform = '../mri/transforms/talairach.xfm'

```

```

>>> parcstats.inputs.brainmask = '../mri/brainmask.mgz'
>>> parcstats.inputs.aseg = '../mri/aseg.presurf.mgz'
>>> parcstats.inputs.ribbon = '../mri/ribbon.mgz'
>>> parcstats.inputs.lh_pial = 'lh.pial'
>>> parcstats.inputs.rh_pial = 'rh.pial'
>>> parcstats.inputs.lh_white = 'lh.white'
>>> parcstats.inputs.rh_white = 'rh.white'
>>> parcstats.inputs.thickness = 'lh.thickness'
>>> parcstats.inputs.surface = 'white'
>>> parcstats.inputs.out_table = 'lh.test.stats'
>>> parcstats.inputs.out_color = 'test.ctab'
>>> parcstats.cmdline
'mris_anatomical_stats -c test.ctab -f lh.test.stats 10335 lh white'

```

Inputs:

```

[Mandatory]
aseg: (an existing file name)
    Input file must be <subject_id>/mri/aseg.presurf.mgz
brainmask: (an existing file name)
    Input file must be <subject_id>/mri/brainmask.mgz
hemisphere: ('lh' or 'rh')
    Hemisphere being processed
    flag: %s, position: -2
lh_pial: (an existing file name)
    Input file must be <subject_id>/surf/lh.pial
lh_white: (an existing file name)
    Input file must be <subject_id>/surf/lh.white
rh_pial: (an existing file name)
    Input file must be <subject_id>/surf/rh.pial
rh_white: (an existing file name)
    Input file must be <subject_id>/surf/rh.white
ribbon: (an existing file name)
    Input file must be <subject_id>/mri/ribbon.mgz
subject_id: (a string, nipy default value: subject_id)
    Subject being processed
    flag: %s, position: -3
thickness: (an existing file name)
    Input file must be <subject_id>/surf/?h.thickness
transform: (an existing file name)
    Input file must be <subject_id>/mri/transforms/talairach.xfm
wm: (an existing file name)
    Input file must be <subject_id>/mri/wm.mgz

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the input
    files to the node directory.
cortex_label: (an existing file name)
    implicit input file {hemi}.cortex.label
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)

```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
in_annotation: (a file name)
    compute properties for each label in the annotation file separately
    flag: -a %s
    mutually_exclusive: in_label
in_cortex: (a file name)
    Input cortex label
    flag: -cortex %s
in_label: (a file name)
    limit calculations to specified label
    flag: -l %s
    mutually_exclusive: in_annotation, out_color
mgz: (a boolean)
    Look for mgz files
    flag: -mgz
out_color: (a file name)
    Output annotation files's colortable to text file
    flag: -c %s
    mutually_exclusive: in_label
out_table: (a file name)
    Table output to tablefile
    flag: -f %s
    requires: tabular_output
subjects_dir: (an existing directory name)
    subjects directory
surface: (a string)
    Input surface (e.g. 'white')
    flag: %s, position: -1
tabular_output: (a boolean)
    Tabular output
    flag: -b
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
th3: (a boolean)
    turns on new vertex-wise volume calc for mris_anat_stats
    flag: -th3
    requires: cortex_label

```

Outputs:

```

out_color: (a file name)
    Output annotation files's colortable to text file
out_table: (a file name)
    Table output to tablefile

```

RelabelHypointensities[Link to code](#)Wraps command **mri_relabel_hypointensities**

Relabel Hypointensities

Examples

```
>>> from nipy.interfaces.freesurfer import RelabelHypointensities
>>> relabelhypos = RelabelHypointensities()
>>> relabelhypos.inputs.lh_white = 'lh.pial'
>>> relabelhypos.inputs.rh_white = 'lh.pial'
>>> relabelhypos.inputs.surf_directory = '.'
>>> relabelhypos.inputs.aseg = 'aseg.mgz'
>>> relabelhypos.cmdline
'mri_relabel_hypointensities aseg.mgz . aseg.hypos.mgz'
```

Inputs:

```
[Mandatory]
aseg: (an existing file name)
      Input aseg file
      flag: %s, position: -3
lh_white: (an existing file name)
          Implicit input file must be lh.white
rh_white: (an existing file name)
          Implicit input file must be rh.white

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          Output aseg file
          flag: %s, position: -1
subjects_dir: (an existing directory name)
              subjects directory
surf_directory: (a directory name, nipy default value: .)
                Directory containing lh.white and rh.white
                flag: %s, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          Output aseg file
          flag: %s
```

RemoveIntersection

[Link to code](#)

Wraps command **mrisc_remove_intersection**

This program removes the intersection of the given MRI

Examples

```
>>> from nipy.interfaces.freesurfer import RemoveIntersection
>>> ri = RemoveIntersection()
>>> ri.inputs.in_file = 'lh.pial'
>>> ri.cmdline
'mris_remove_intersection lh.pial lh.pial'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Input file for RemoveIntersection
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          Output file for RemoveIntersection
          flag: %s, position: -1
subjects_dir: (an existing directory name)
              subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          Output file for RemoveIntersection
```

RemoveNeck

[Link to code](#)

Wraps command **mri_remove_neck**

Crops the neck out of the mri image

Examples

```
>>> from nipy.interfaces.freesurfer import TalairachQC
>>> remove_neck = RemoveNeck()
>>> remove_neck.inputs.in_file = 'norm.mgz'
>>> remove_neck.inputs.transform = 'trans.mat'
>>> remove_neck.inputs.template = 'trans.mat'
>>> remove_neck.cmdline
'mri_remove_neck norm.mgz trans.mat trans.mat norm_noneck.mgz'
```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    Input file for RemoveNeck
    flag: %s, position: -4
template: (an existing file name)
    Input template file for RemoveNeck
    flag: %s, position: -2
transform: (an existing file name)
    Input transform file for RemoveNeck
    flag: %s, position: -3

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    Output file for RemoveNeck
    flag: %s, position: -1
radius: (an integer (int or long))
    Radius
    flag: -radius %d
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
    Output file with neck removed

```

SampleToSurface[Link to code](#)**Wraps command `mri_vol2surf`**Sample a volume to the cortical surface using Freesurfer's `mri_vol2surf`.

You must supply a sampling method, range, and units. You can project either a given distance (in mm) or a given fraction of the cortical thickness at that vertex along the surface normal from the target surface, and then set the value of that vertex to be either the value at that point or the average or maximum value found along the projection vector.

By default, the surface will be saved as a vector with a length equal to the number of vertices on the target surface. This is not a problem for Freesurfer programs, but if you intend to use the file with interfaces to another package, you must set the `reshape` input to `True`, which will factor the surface vector into a matrix with dimensions compatible with proper Nifti files.

Examples

```
>>> import nipy.interfaces.freesurfer as fs
>>> sampler = fs.SampleToSurface(hemi="lh")
>>> sampler.inputs.source_file = "copel.nii.gz"
>>> sampler.inputs.reg_file = "register.dat"
>>> sampler.inputs.sampling_method = "average"
>>> sampler.inputs.sampling_range = 1
>>> sampler.inputs.sampling_units = "frac"
>>> sampler.cmdline
'mri_vol2surf --hemi lh --o ...lh.copel.mgz --reg register.dat --projfrac-avg 1.
 000 --mov copel.nii.gz'
>>> res = sampler.run()
```

Inputs:

```
[Mandatory]
hemi: ('lh' or 'rh')
    target hemisphere
    flag: --hemi %s
mni152reg: (a boolean)
    source volume is in MNI152 space
    flag: --mni152reg
    mutually_exclusive: reg_file, reg_header, mni152reg
projection_stem: (a string)
    stem for precomputed linear estimates and volume fractions
    mutually_exclusive: sampling_method
reg_file: (an existing file name)
    source-to-reference registration file
    flag: --reg %s
    mutually_exclusive: reg_file, reg_header, mni152reg
reg_header: (a boolean)
    register based on header geometry
    flag: --regheader %s
    mutually_exclusive: reg_file, reg_header, mni152reg
    requires: subject_id
sampling_method: ('point' or 'max' or 'average')
    how to sample -- at a point or at the max or average over a range
    flag: %s
    mutually_exclusive: projection_stem
    requires: sampling_range, sampling_units
source_file: (an existing file name)
    volume to sample values from
    flag: --mov %s

[Optional]
apply_rot: (a tuple of the form: (a float, a float, a float))
    rotation angles (in degrees) to apply to reg matrix
    flag: --rot %.3f %.3f %.3f
apply_trans: (a tuple of the form: (a float, a float, a float))
    translation (in mm) to apply to reg matrix
    flag: --trans %.3f %.3f %.3f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cortex_mask: (a boolean)
    mask the target surface with hemi.cortex.label
    flag: --cortex
    mutually_exclusive: mask_label
```

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
fix_tk_reg: (a boolean)
         make reg matrix round-compatible
         flag: --fixtkreg
float2int_method: ('round' or 'tkregister')
         method to convert reg matrix values (default is round)
         flag: --float2int %s
frame: (an integer (int or long))
         save only one frame (0-based)
         flag: --frame %d
hits_file: (a boolean or an existing file name)
         save image with number of hits at each voxel
         flag: --srchit %s
hits_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
           'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or
           'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti1' or 'nii' or
           'niigz')
         hits file type
         flag: --srchit_type
ico_order: (an integer (int or long))
         icosahedron order when target_subject is 'ico'
         flag: --icoorder %d
         requires: target_subject
ignore_exception: (a boolean, nipy default value: False)
         Print an error message instead of throwing an exception in case the
         interface fails to run
interp_method: ('nearest' or 'trilinear')
         interpolation method
         flag: --interp %s
mask_label: (an existing file name)
         label file to mask output with
         flag: --mask %s
         mutually_exclusive: cortex_mask
no_reshape: (a boolean)
         do not reshape surface vector (default)
         flag: --noreshape
         mutually_exclusive: reshape
out_file: (a file name)
         surface file to write
         flag: --o %s
out_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
          'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or
          'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti1' or 'nii' or
          'niigz' or 'gii')
         output file type
         flag: --out_type %s
override_reg_subj: (a boolean)
         override the subject in the reg file header
         flag: --srcsubject %s
         requires: subject_id
reference_file: (an existing file name)
         reference volume (default is orig.mgz)
         flag: --ref %s
reshape: (a boolean)
         reshape surface vector to fit in non-mgh format

```

```

    flag: --reshape
    mutually_exclusive: no_reshape
reshape_slices: (an integer (int or long))
    number of 'slices' for reshaping
    flag: --rf %d
sampling_range: (a float or a tuple of the form: (a float, a float, a
    float))
    sampling range - a point or a tuple of (min, max, step)
sampling_units: ('mm' or 'frac')
    sampling range type -- either 'mm' or 'frac'
scale_input: (a float)
    multiple all intensities by scale factor
    flag: --scale %.3f
smooth_surf: (a float)
    smooth output surface (mm fwhm)
    flag: --surf-fwhm %.3f
smooth_vol: (a float)
    smooth input volume (mm fwhm)
    flag: --fwhm %.3f
subject_id: (a string)
    subject id
subjects_dir: (an existing directory name)
    subjects directory
surf_reg: (a boolean)
    use surface registration to target subject
    flag: --surfreg
    requires: target_subject
surface: (a string)
    target surface (default is white)
    flag: --surf %s
target_subject: (a string)
    sample to surface of different subject than source
    flag: --trgsurface %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
vox_file: (a boolean or a file name)
    text file with the number of voxels intersecting the surface
    flag: --nvox %s

```

Outputs:

```

hits_file: (an existing file name)
    image with number of hits at each voxel
out_file: (an existing file name)
    surface file
vox_file: (an existing file name)
    text file with the number of voxels intersecting the surface

```

SmoothTessellation[Link to code](#)Wraps command **mrissmooth**

This program smooths the tessellation of a surface using 'mrissmooth'

See also:**SurfaceSmooth() Interface** For smoothing a scalar field along a surface manifold

Example

```
>>> import nipy.interfaces.freesurfer as fs
>>> smooth = fs.SmoothTessellation()
>>> smooth.inputs.in_file = 'lh.hippocampus.stl'
>>> smooth.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Input volume to tessellate voxels from.
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
curvature_averaging_iterations: (an integer (int or long))
      Number of curvature averaging iterations (default=10)
      flag: -a %d
disable_estimates: (a boolean)
      Disables the writing of curvature and area estimates
      flag: -nw
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
gaussian_curvature_norm_steps: (an integer (int or long))
      Use Gaussian curvature smoothing
      flag: %d
gaussian_curvature_smoothing_steps: (an integer (int or long))
      Use Gaussian curvature smoothing
      flag: %d
ignore_exception: (a boolean, nipy default value: False)
      Print an error message instead of throwing an exception in case the
      interface fails to run
normalize_area: (a boolean)
      Normalizes the area after smoothing
      flag: -area
out_area_file: (a file name)
      Write area to ?h.areaname (default "area")
      flag: -b %s
out_curvature_file: (a file name)
      Write curvature to ?h.curvname (default "curv")
      flag: -c %s
out_file: (a file name)
      output filename or True to generate one
      flag: %s, position: -1
seed: (an integer (int or long))
      Seed for setting random number generator
      flag: -seed %d
smoothing_iterations: (an integer (int or long))
      Number of smoothing iterations (default=10)
      flag: -n %d
snapshot_writing_iterations: (an integer (int or long))
      Write snapshot every "n" iterations
      flag: -w %d
subjects_dir: (an existing directory name)
```

```

    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
use_gaussian_curvature_smoothing: (a boolean)
    Use Gaussian curvature smoothing
    flag: -g
use_momentum: (a boolean)
    Uses momentum
    flag: -m

```

Outputs:

```

surface: (an existing file name)
    Smoothed surface file

```

Sphere

[Link to code](#)Wraps command **mr**is_sphere

This program will add a template into an average surface

Examples

```

>>> from nipy.interfaces.freesurfer import Sphere
>>> sphere = Sphere()
>>> sphere.inputs.in_file = 'lh.pial'
>>> sphere.cmdline
'mris_sphere lh.pial lh.sphere'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    Input file for Sphere
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_smoothwm: (an existing file name)
    Input surface required when -q flag is not selected
magic: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    flag: -q
num_threads: (an integer (int or long))
    allows for specifying more threads

```

```

out_file: (a file name)
    Output file for Sphere
    flag: %s, position: -1
seed: (an integer (int or long))
    Seed for setting random number generator
    flag: -seed %d
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
    Output file for Sphere

```

Surface2VolTransform[Link to code](#)Wraps command **mri_surf2vol**

Use FreeSurfer mri_surf2vol to apply a transform.

Examples

```

>>> from nipy.interfaces.freesurfer import Surface2VolTransform
>>> xfm2vol = Surface2VolTransform()
>>> xfm2vol.inputs.source_file = 'lh.cope1.mgz'
>>> xfm2vol.inputs.reg_file = 'register.mat'
>>> xfm2vol.inputs.hemi = 'lh'
>>> xfm2vol.inputs.template_file = 'cope1.nii.gz'
>>> xfm2vol.inputs.subjects_dir = '.'
>>> xfm2vol.cmdline
'mri_surf2vol --hemi lh --volreg register.mat --surfval lh.cope1.mgz --sd . --
↳template cope1.nii.gz --outvol lh.cope1_asVol.nii --vtxvol lh.cope1_asVol_
↳vertex.nii'
>>> res = xfm2vol.run()

```

Inputs:

```

[Mandatory]
hemi: (a unicode string)
    hemisphere of data
    flag: --hemi %s
reg_file: (an existing file name)
    tkRAS-to-tkRAS matrix (tkregister2 format)
    flag: --volreg %s
    mutually_exclusive: subject_id
source_file: (an existing file name)
    This is the source of the surface values
    flag: --surfval %s
    mutually_exclusive: mkmask

[Optional]
args: (a unicode string)
    Additional parameters to the command

```

```

    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mkmask: (a boolean)
    make a mask instead of loading surface values
    flag: --mkmask
    mutually_exclusive: source_file
projfrac: (a float)
    thickness fraction
    flag: --projfrac %s
subject_id: (a unicode string)
    subject id
    flag: --identity %s
    mutually_exclusive: reg_file
subjects_dir: (a unicode string)
    freesurfer subjects directory defaults to $SUBJECTS_DIR
    flag: --sd %s
surf_name: (a unicode string)
    surfname (default is white)
    flag: --surf %s
template_file: (an existing file name)
    Output template volume
    flag: --template %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformed_file: (a file name)
    Output volume
    flag: --outvol %s
vertexvol_file: (a file name)
    Path name of the vertex output volume, which is the same as output
    volume except that the value of each voxel is the vertex-id that is
    mapped to that voxel.
    flag: --vtxvol %s

```

Outputs:

```

transformed_file: (an existing file name)
    Path to output file if used normally
vertexvol_file: (a file name)
    vertex map volume path id. Optional

```

SurfaceSmooth[Link to code](#)**Wraps command `mri_surf2surf`**Smooth a surface image with `mri_surf2surf`.

The surface is smoothed by an iterative process of averaging the value at each vertex with those of its adjacent neighbors. You may supply either the number of iterations to run or a desired effective FWHM of the smoothing process. If the latter, the underlying program will calculate the correct number of iterations internally.

See also:

SmoothTessellation() Interface For smoothing a tessellated surface (e.g. in gifti or .stl)

Examples

```
>>> import nipyne.interfaces.freesurfer as fs
>>> smoother = fs.SurfaceSmooth()
>>> smoother.inputs.in_file = "lh.cope1.mgz"
>>> smoother.inputs.subject_id = "subj_1"
>>> smoother.inputs.hemi = "lh"
>>> smoother.inputs.fwhm = 5
>>> smoother.cmdline
'mri_surf2surf --cortex --fwhm 5.0000 --hemi lh --sval lh.cope1.mgz --tval ...lh.
↳ cope1_smooth5.mgz --s subj_1'
>>> smoother.run()
```

Inputs:

```
[Mandatory]
hemi: ('lh' or 'rh')
    hemisphere to operate on
    flag: --hemi %s
in_file: (a file name)
    source surface file
    flag: --sval %s
subject_id: (a string)
    subject id of surface file
    flag: --s %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cortex: (a boolean, nipyne default value: True)
    only smooth within $hemi.cortex.label
    flag: --cortex
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
fwhm: (a float)
    effective FWHM of the smoothing process
    flag: --fwhm %.4f
    mutually_exclusive: smooth_iters
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    surface file to write
    flag: --tval %s
reshape: (a boolean)
    reshape surface vector to fit in non-mgh format
    flag: --reshape
smooth_iters: (an integer (int or long))
    iterations of the smoothing process
    flag: --smooth %d
    mutually_exclusive: fwhm
subjects_dir: (an existing directory name)
    subjects directory
```



```
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    smoothed surface file
```

SurfaceSnapshots

[Link to code](#)**Wraps command `tksurfer`**

Use Tksurfer to save pictures of the cortical surface.

By default, this takes snapshots of the lateral, medial, ventral, and dorsal surfaces. See the `six_images` option to add the anterior and posterior surfaces.You may also supply your own tcl script (see the [Freesurfer wiki](#) for information on scripting tksurfer). The screenshot stem is set as the environment variable “_SNAPSHOT_STEM”, which you can use in your own scripts.

Node that this interface will not run if you do not have graphics enabled on your system.

Examples

```
>>> import nipy.interfaces.freesurfer as fs
>>> shots = fs.SurfaceSnapshots(subject_id="fsaverage", hemi="lh", surface="pial")
>>> shots.inputs.overlay = "zstat1.nii.gz"
>>> shots.inputs.overlay_range = (2.3, 6)
>>> shots.inputs.overlay_reg = "register.dat"
>>> res = shots.run()
```

Inputs:

```
[Mandatory]
hemi: ('lh' or 'rh')
    hemisphere to visualize
    flag: %s, position: 2
subject_id: (a string)
    subject to visualize
    flag: %s, position: 1
surface: (a string)
    surface to visualize
    flag: %s, position: 3

[Optional]
annot_file: (an existing file name)
    path to annotation file to display
    flag: -annotation %s
    mutually_exclusive: annot_name
annot_name: (a string)
    name of annotation to display (must be in $subject/label directory
    flag: -annotation %s
    mutually_exclusive: annot_file
args: (a unicode string)
    Additional parameters to the command
    flag: %s
colortable: (an existing file name)
```

```

    load colortable file
    flag: -colortable %s
demean_overlay: (a boolean)
    remove mean from overlay
    flag: -zm
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
identity_reg: (a boolean)
    use the identity matrix to register the overlay to the surface
    flag: -overlay-reg-identity
    mutually_exclusive: overlay_reg, identity_reg, mni152_reg
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert_overlay: (a boolean)
    invert the overlay display
    flag: -invphaseflag 1
label_file: (an existing file name)
    path to label file to display
    flag: -label %s
    mutually_exclusive: label_name
label_name: (a string)
    name of label to display (must be in $subject/label directory)
    flag: -label %s
    mutually_exclusive: label_file
label_outline: (a boolean)
    draw label/annotation as outline
    flag: -label-outline
label_under: (a boolean)
    draw label/annotation under overlay
    flag: -labels-under
mni152_reg: (a boolean)
    use to display a volume in MNI152 space on the average subject
    flag: -mni152reg
    mutually_exclusive: overlay_reg, identity_reg, mni152_reg
orig_suffix: (a string)
    set the orig surface suffix string
    flag: -orig %s
overlay: (an existing file name)
    load an overlay volume/surface
    flag: -overlay %s
    requires: overlay_range
overlay_range: (a float or a tuple of the form: (a float, a float) or
    a tuple of the form: (a float, a float, a float))
    overlay range--either min, (min, max) or (min, mid, max)
    flag: %s
overlay_range_offset: (a float)
    overlay range will be symetric around offset value
    flag: -offset %.3f
overlay_reg: (a file name)
    registration matrix file to register overlay to surface
    flag: -overlay-reg %s
    mutually_exclusive: overlay_reg, identity_reg, mni152_reg
patch_file: (an existing file name)
    load a patch
    flag: -patch %s

```

```

reverse_overlay: (a boolean)
    reverse the overlay display
    flag: -revphaseflag 1
screenshot_stem: (a string)
    stem to use for screenshot file names
show_color_scale: (a boolean)
    display the color scale bar
    flag: -colscalebarflag 1
show_color_text: (a boolean)
    display text in the color scale bar
    flag: -colscaletext 1
show_curv: (a boolean)
    show curvature
    flag: -curv
    mutually_exclusive: show_gray_curv
show_gray_curv: (a boolean)
    show curvature in gray
    flag: -gray
    mutually_exclusive: show_curv
six_images: (a boolean)
    also take anterior and posterior snapshots
sphere_suffix: (a string)
    set the sphere.reg suffix string
    flag: -sphere %s
stem_template_args: (a list of items which are a string)
    input names to use as arguments for a string-formated stem template
    requires: screenshot_stem
subjects_dir: (an existing directory name)
    subjects directory
tcl_script: (an existing file name)
    override default screenshot script
    flag: %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
truncate_overlay: (a boolean)
    truncate the overlay display
    flag: -truncphaseflag 1

```

Outputs:

```

snapshots: (a list of items which are an existing file name)
    tiff images of the surface from different perspectives

```

SurfaceTransform[Link to code](#)**Wraps command `mri_surf2surf`**

Transform a surface file from one subject to another via a spherical registration.

Both the source and target subject must reside in your Subjects Directory, and they must have been processed with recon-all, unless you are transforming to one of the icosahedron meshes.

Examples

```

>>> from nipy.interfaces.freesurfer import SurfaceTransform
>>> sxfm = SurfaceTransform()

```

```
>>> sxfm.inputs.source_file = "lh.copel.nii.gz"
>>> sxfm.inputs.source_subject = "my_subject"
>>> sxfm.inputs.target_subject = "fsaverage"
>>> sxfm.inputs.hemi = "lh"
>>> sxfm.run()
```

Inputs:

```
[Mandatory]
hemi: ('lh' or 'rh')
    hemisphere to transform
    flag: --hemi %s
source_annot_file: (an existing file name)
    surface annotation file
    flag: --sval-annot %s
    mutually_exclusive: source_file
source_file: (an existing file name)
    surface file with source values
    flag: --sval %s
    mutually_exclusive: source_annot_file
source_subject: (a string)
    subject id for source surface
    flag: --srcsubject %s
target_subject: (a string)
    subject id of target surface
    flag: --trgsurface %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    surface file to write
    flag: --tval %s
reshape: (a boolean)
    reshape output surface to conform with Nifti
    flag: --reshape
reshape_factor: (an integer (int or long))
    number of slices in reshaped image
    flag: --reshape-factor
source_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
    'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or
    'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti' or 'nii' or
    'niigz')
    source file format
    flag: --sfmt %s
    requires: source_file
subjects_dir: (an existing directory name)
    subjects directory
target_ico_order: (1 or 2 or 3 or 4 or 5 or 6 or 7)
    order of the icosahedron if target_subject is 'ico'
```

```

        flag: --trgicoorder %d
target_type: ('cor' or 'mgh' or 'mgz' or 'minc' or 'analyze' or
              'analyze4d' or 'spm' or 'afni' or 'brik' or 'bshort' or 'bfloat' or
              'sdt' or 'outline' or 'otl' or 'gdf' or 'nifti1' or 'nii' or
              'niigz' or 'gii')
        output format
        flag: --tfmt %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
          transformed surface file

```

TalairachAVI[Link to code](#)Wraps command **talairach_avi**

Front-end for Avi Snyders image registration tool. Computes the talairach transform that maps the input volume to the MNI average_305. This does not add the xfm to the header of the input file. When called by recon-all, the xfm is added to the header after the transform is computed.

Examples

```

>>> from nipy.interfaces.freesurfer import TalairachAVI
>>> example = TalairachAVI()
>>> example.inputs.in_file = 'norm.mgz'
>>> example.inputs.out_file = 'trans.mat'
>>> example.cmdline
'talairach_avi --i norm.mgz --xfm trans.mat'

```

```

>>> example.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         input volume
         flag: --i %s
out_file: (a file name)
          output xfm file
          flag: --xfm %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
atlas: (a string)
       alternate target atlas (in freesurfer/average dir)
       flag: --atlas %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables

```

```
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
    The output transform for TalairachAVI
out_log: (a file name)
    The output log file for TalairachAVI
out_txt: (a file name)
    The output text file for TaliarachAVI
```

TalairachQC

[Link to code](#)Wraps command **tal_QC_AZS**

Examples

```
>>> from nipy.interfaces.freesurfer import TalairachQC
>>> qc = TalairachQC()
>>> qc.inputs.log_file = 'dirs.txt'
>>> qc.cmdline
'tal_QC_AZS dirs.txt'
```

Inputs:

```
[Mandatory]
log_file: (an existing file name)
    The log file for TalairachQC
    flag: %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
log_file: (an existing file name, nipy default value:
          stdout.nipy)
          The output log
```

Tkregister2

[Link to code](#)

Wraps command **tkregister2**

Examples

Get transform matrix between orig (*tkRAS*) and native (*scannerRAS*) coordinates in Freesurfer. Implements the first step of mapping surfaces to native space in [this guide](#).

```
>>> from nipy.interfaces.freesurfer import Tkregister2
>>> tk2 = Tkregister2(reg_file='T1_to_native.dat')
>>> tk2.inputs.moving_image = 'T1.mgz'
>>> tk2.inputs.target_image = 'structural.nii'
>>> tk2.inputs.reg_header = True
>>> tk2.cmdline
'tkregister2 --mov T1.mgz --noedit --reg T1_to_native.dat --regheader --targ_
↳ structural.nii'
>>> tk2.run()
```

The example below uses tkregister2 without the manual editing stage to convert FSL-style registration matrix (.mat) to FreeSurfer-style registration matrix (.dat)

```
>>> from nipy.interfaces.freesurfer import Tkregister2
>>> tk2 = Tkregister2()
>>> tk2.inputs.moving_image = 'epi.nii'
>>> tk2.inputs.fsl_in_matrix = 'flirt.mat'
>>> tk2.cmdline
'tkregister2 --fsl flirt.mat --mov epi.nii --noedit --reg register.dat'
>>> tk2.run()
```

Inputs:

```
[Mandatory]
moving_image: (an existing file name)
               moving volume
               flag: --mov %s
reg_file: (a file name, nipy default value: register.dat)
           freesurfer-style registration file
           flag: --reg %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
fsl_in_matrix: (an existing file name)
               fsl-style registration input matrix
               flag: --fsl %s
fsl_out: (a file name)
          compute an FSL-compatible resgitration matrix
```

```

    flag: --fslregout %s
fstal: (a boolean)
    set mov to be tal and reg to be tal xfm
    flag: --fstal
    mutually_exclusive: target_image, moving_image
fstarg: (a boolean)
    use subject's T1 as reference
    flag: --fstarg
    mutually_exclusive: target_image
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
movscale: (a float)
    adjust registration matrix to scale mov
    flag: --movscale %f
noedit: (a boolean, nipy default value: True)
    do not open edit window (exit)
    flag: --noedit
reg_header: (a boolean)
    compute registration from headers
    flag: --regheader
subject_id: (a string)
    freesurfer subject ID
    flag: --s %s
subjects_dir: (an existing directory name)
    subjects directory
target_image: (an existing file name)
    target volume
    flag: --targ %s
    mutually_exclusive: fstarg
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xfm: (an existing file name)
    use a matrix in MNI coordinates as initial registration
    flag: --xfm %s

```

Outputs:

```

fsl_file: (a file name)
    FSL-style registration file
reg_file: (an existing file name)
    freesurfer-style registration file

```

VolumeMask[Link to code](#)Wraps command **mrisc_volmask**

Computes a volume mask, at the same resolution as the <subject>/mri/brain.mgz. The volume mask contains 4 values: LH_WM (default 10), LH_GM (default 100), RH_WM (default 20), RH_GM (default 200). The algorithm uses the 4 surfaces situated in <subject>/surf/ [lh|rh].[white|pial] and labels voxels based on the signed-distance function from the surface.

Examples

```
>>> from nipype.interfaces.freesurfer import VolumeMask
>>> volmask = VolumeMask()
>>> volmask.inputs.left_whitelabel = 2
>>> volmask.inputs.left_ribbonlabel = 3
>>> volmask.inputs.right_whitelabel = 41
>>> volmask.inputs.right_ribbonlabel = 42
>>> volmask.inputs.lh_pial = 'lh.pial'
>>> volmask.inputs.rh_pial = 'lh.pial'
>>> volmask.inputs.lh_white = 'lh.pial'
>>> volmask.inputs.rh_white = 'lh.pial'
>>> volmask.inputs.subject_id = '10335'
>>> volmask.inputs.save_ribbon = True
>>> volmask.cmdline
'mris_volmask --label_left_ribbon 3 --label_left_white 2 --label_right_ribbon 42 -
↪-label_right_white 41 --save_ribbon 10335'
```

Inputs:

```
[Mandatory]
left_ribbonlabel: (an integer (int or long))
    Left cortical ribbon label
    flag: --label_left_ribbon %d
left_whitelabel: (an integer (int or long))
    Left white matter label
    flag: --label_left_white %d
lh_pial: (an existing file name)
    Implicit input left pial surface
lh_white: (an existing file name)
    Implicit input left white matter surface
rh_pial: (an existing file name)
    Implicit input right pial surface
rh_white: (an existing file name)
    Implicit input right white matter surface
right_ribbonlabel: (an integer (int or long))
    Right cortical ribbon label
    flag: --label_right_ribbon %d
right_whitelabel: (an integer (int or long))
    Right white matter label
    flag: --label_right_white %d
subject_id: (a string, nipype default value: subject_id)
    Subject being processed
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
aseg: (an existing file name)
    Implicit aseg.mgz segmentation. Specify a different aseg by using
    the 'in_aseg' input.
    mutually_exclusive: in_aseg
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the implicit
    input files to the node directory.
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
```

```
Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_aseg: (an existing file name)
    Input aseg file for VolumeMask
    flag: --aseg_name %s
    mutually_exclusive: aseg
save_ribbon: (a boolean)
    option to save just the ribbon for the hemispheres in the format
    ?h.ribbon.mgz
    flag: --save_ribbon
subjects_dir: (an existing directory name)
    subjects directory
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
lh_ribbon: (a file name)
    Output left cortical ribbon mask
out_ribbon: (a file name)
    Output cortical ribbon mask
rh_ribbon: (a file name)
    Output right cortical ribbon mask
```

copy2subjdir()[Link to code](#)

Method to copy an input to the subjects directory

createoutputdirs()[Link to code](#)

create all output directories. If not created, some freesurfer interfaces fail

interfaces.fsl.ICA_AROMA

ICA_AROMA

[Link to code](#)

Wraps command **ICA_AROMA.py**

Interface for the ICA_AROMA.py script.

ICA-AROMA (i.e. ‘ICA-based Automatic Removal Of Motion Artifacts’) concerns a data-driven method to identify and remove motion-related independent components from fMRI data. To that end it exploits a small, but robust set of theoretically motivated features, preventing the need for classifier re-training and therefore providing direct and easy applicability.

See link for further documentation: <https://github.com/rhr-pruim/ICA-AROMA>

Example

```
>>> from nipy.interfaces.fsl import ICA_AROMA
>>> from nipy.testing import example_data
>>> AROMA_obj = ICA_AROMA.ICA_AROMA()
>>> AROMA_obj.inputs.in_file = 'functional.nii'
>>> AROMA_obj.inputs.mat_file = 'func_to_struct.mat'
>>> AROMA_obj.inputs.fnirt_warp_file = 'warpfield.nii'
>>> AROMA_obj.inputs.motion_parameters = 'fsl_mcflirt_movpar.txt'
>>> AROMA_obj.inputs.mask = 'mask.nii.gz'
>>> AROMA_obj.inputs.denoise_type = 'both'
>>> AROMA_obj.inputs.out_dir = 'ICA_testout'
>>> AROMA_obj.cmdline
'ICA_AROMA.py -den both -warp warpfield.nii -i functional.nii -m mask.nii.gz -
↪affmat func_to_struct.mat -mc fsl_mcflirt_movpar.txt -o ICA_testout'
```

Inputs:

```
[Mandatory]
denoise_type: ('nonaggr' or 'aggr' or 'both' or 'no', nipy default
               value: nonaggr)
               Type of denoising strategy:
               -none: only classification, no denoising
```

```

    -nonaggr (default): non-aggresssive denoising, i.e. partial
    component regression
    -aggr: aggressive denoising, i.e. full component regression
    -both: both aggressive and non-aggressive denoising (two outputs)
    flag: -den %s
feat_dir: (an existing directory name)
    If a feat directory exists and temporal filtering has not been run
    yet, ICA_AROMA can use the files in this directory.
    flag: -feat %s
    mutually_exclusive: in_file, mat_file, fnirt_warp_file,
    motion_parameters
in_file: (an existing file name)
    volume to be denoised
    flag: -i %s
    mutually_exclusive: feat_dir
motion_parameters: (an existing file name)
    motion parameters file
    flag: -mc %s
    mutually_exclusive: feat_dir
out_dir: (a directory name)
    output directory
    flag: -o %s

[Optional]
TR: (a float)
    TR in seconds. If this is not specified the TR will be extracted
    from the header of the fMRI nifti file.
    flag: -tr %.3f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dim: (an integer (int or long))
    Dimensionality reduction when running MELODIC (default is automatic
    estimation)
    flag: -dim %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fnirt_warp_file: (an existing file name)
    File name of the warp-file describing the non-linear registration
    (e.g. FSL FNIRT) of the structural data to MNI152 space (.nii.gz)
    flag: -warp %s
    mutually_exclusive: feat_dir
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    path/name volume mask
    flag: -m %s
    mutually_exclusive: feat_dir
mat_file: (an existing file name)
    path/name of the mat-file describing the affine registration (e.g.
    FSL FLIRT) of the functional data to structural space (.mat file)
    flag: -affmat %s
    mutually_exclusive: feat_dir
melodic_dir: (an existing directory name)
    path to MELODIC directory if MELODIC has already been run

```

```

    flag: -meldir %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

aggr_denoised_file: (an existing file name)
    if generated: aggressively denoised volume
nonaggr_denoised_file: (an existing file name)
    if generated: non aggressively denoised volume
out_dir: (an existing directory name)
    directory contains (in addition to the denoised files): melodic.ica
    + classified_motion_components + classification_overview +
    feature_scores + melodic_ic_mni)

```

interfaces.fsl.dti**BEDPOSTX5**[Link to code](#)Wraps command **bedpostx**

BEDPOSTX stands for Bayesian Estimation of Diffusion Parameters Obtained using Sampling Techniques. The X stands for modelling Crossing Fibres. bedpostx runs Markov Chain Monte Carlo sampling to build up distributions on diffusion parameters at each voxel. It creates all the files necessary for running probabilistic tractography. For an overview of the modelling carried out within bedpostx see this [technical report](#).

Note: Consider using `nipy.workflows.fsl.dmri.create_bedpostx_pipeline()` instead.

Example

```

>>> from nipy.interfaces import fsl
>>> bedp = fsl.BEDPOSTX5(bvecs='bvecs', bvals='bvals', dwi='diffusion.nii',
...                      mask='mask.nii', n_fibres=1)
>>> bedp.cmdline
'bedpostx bedpostx --forcedir -n 1'

```

Inputs:

```

[Mandatory]
bvals: (an existing file name)
    b values file
bvecs: (an existing file name)
    b vectors file
dwi: (an existing file name)
    diffusion weighted image data file
mask: (an existing file name)
    bet binary mask file
n_fibres: (a long integer >= 1, nipy default value: 1)
    Maximum number of fibres to fit in each voxel
    flag: -n %d
out_dir: (a directory name, nipy default value: bedpostx)
    output directory
    flag: %s, position: 1

```

```
[Optional]
all_ard: (a boolean)
    Turn ARD on on all fibres
    flag: --allard
    mutually_exclusive: no_ard, all_ard
args: (a unicode string)
    Additional parameters to the command
    flag: %s
burn_in: (a long integer >= 0)
    Total num of jumps at start of MCMC to be discarded
    flag: -b %d
burn_in_no_ard: (a long integer >= 0)
    num of burnin jumps before the ard is imposed
    flag: --burninnoard=%d
cnlinear: (a boolean)
    Initialise with constrained nonlinear fitting
    flag: --cnlinear
    mutually_exclusive: no_spat, non_linear, cnlinear
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
f0_ard: (a boolean)
    Noise floor model: add to the model an unattenuated signal
    compartment f0
    flag: --f0 --ardf0
    mutually_exclusive: f0_noard, f0_ard, all_ard
f0_noard: (a boolean)
    Noise floor model: add to the model an unattenuated signal
    compartment f0
    flag: --f0
    mutually_exclusive: f0_noard, f0_ard
force_dir: (a boolean, nipyne default value: True)
    use the actual directory name given (do not add + to make a new
    directory)
    flag: --forcedir
fudge: (an integer (int or long))
    ARD fudge factor
    flag: -w %d
grad_dev: (an existing file name)
    grad_dev file, if gradnonlin, -g is True
gradnonlin: (a boolean)
    consider gradient nonlinearities, default off
    flag: -g
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
logdir: (a directory name)
    flag: --logdir=%s
model: (1 or 2 or 3)
    use monoexponential (1, default, required for single-shell) or
    multiexponential (2, multi-shell) model
    flag: -model %d
n_jumps: (an integer (int or long))
    Num of jumps to be made by MCMC
    flag: -j %d
no_ard: (a boolean)
```

```

    Turn ARD off on all fibres
    flag: --noard
    mutually_exclusive: no_ard, all_ard
no_spat: (a boolean)
    Initialise with tensor, not spatially
    flag: --nospat
    mutually_exclusive: no_spat, non_linear, cnlinear
non_linear: (a boolean)
    Initialise with nonlinear fitting
    flag: --nonlinear
    mutually_exclusive: no_spat, non_linear, cnlinear
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
    FSL output type
rician: (a boolean)
    use Rician noise modeling
    flag: --rician
sample_every: (a long integer >= 0)
    Num of jumps for each sample (MCMC)
    flag: -s %d
seed: (an integer (int or long))
    seed for pseudo random number generator
    flag: --seed=%d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
update_proposal_every: (a long integer >= 1)
    Num of jumps for each update to the proposal density std (MCMC)
    flag: --updateproposalevery=%d
use_gpu: (a boolean)
    Use the GPU version of bedpostx

```

Outputs:

```

dyads: (a list of items which are an existing file name)
    Mean of PDD distribution in vector form.
dyads_dispersion: (a list of items which are an existing file name)
    Dispersion
mean_S0samples: (an existing file name)
    Mean of distribution on T2wbaseline signal intensity S0
mean_dsamples: (an existing file name)
    Mean of distribution on diffusivity d
mean_fsamples: (a list of items which are an existing file name)
    Mean of distribution on f anisotropy
mean_phsamples: (a list of items which are an existing file name)
    Mean of distribution on phi
mean_thsamples: (a list of items which are an existing file name)
    Mean of distribution on theta
merged_fsamples: (a list of items which are an existing file name)
    Samples from the distribution on anisotropic volume fraction
merged_phsamples: (a list of items which are an existing file name)
    Samples from the distribution on phi
merged_thsamples: (a list of items which are an existing file name)
    Samples from the distribution on theta

```

References:: None

DTIFit

[Link to code](#)

Wraps command **dtifit**

Use FSL dtifit command for fitting a diffusion tensor model at each voxel

Example

```
>>> from nipy.interfaces import fsl
>>> dti = fsl.DTIFit()
>>> dti.inputs.dwi = 'diffusion.nii'
>>> dti.inputs.bvecs = 'bvecs'
>>> dti.inputs.bvals = 'bvals'
>>> dti.inputs.base_name = 'TP'
>>> dti.inputs.mask = 'mask.nii'
>>> dti.cmdline
'dtifit -k diffusion.nii -o TP -m mask.nii -r bvecs -b bvals'
```

Inputs:

```
[Mandatory]
bvals: (an existing file name)
      b values file
      flag: -b %s, position: 4
bvecs: (an existing file name)
      b vectors file
      flag: -r %s, position: 3
dwi: (an existing file name)
     diffusion weighted image data file
     flag: -k %s, position: 0
mask: (an existing file name)
     bet binary mask file
     flag: -m %s, position: 2

[Optional]
args: (a unicode string)
     Additional parameters to the command
     flag: %s
base_name: (a unicode string, nipy default value: dtifit_)
     base_name that all output files will start with
     flag: -o %s, position: 1
cni: (an existing file name)
     input counfound regressors
     flag: --cni=%s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
gradnonlin: (an existing file name)
           gradient non linearities
           flag: --gradnonlin=%s
ignore_exception: (a boolean, nipy default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
little_bit: (a boolean)
            only process small area of brain
            flag: --littlebit
max_x: (an integer (int or long))
```



```

        max x
        flag: -X %d
max_y: (an integer (int or long))
        max y
        flag: -Y %d
max_z: (an integer (int or long))
        max z
        flag: -Z %d
min_x: (an integer (int or long))
        min x
        flag: -x %d
min_y: (an integer (int or long))
        min y
        flag: -y %d
min_z: (an integer (int or long))
        min z
        flag: -z %d
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
            FSL output type
save_tensor: (a boolean)
            save the elements of the tensor
            flag: --save_tensor
sse: (a boolean)
            output sum of squared errors
            flag: --sse
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
            Control terminal output: `stream` - displays to terminal immediately
            (default), `allatonce` - waits till command is finished to display
            output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

FA: (an existing file name)
    path/name of file with the fractional anisotropy
L1: (an existing file name)
    path/name of file with the 1st eigenvalue
L2: (an existing file name)
    path/name of file with the 2nd eigenvalue
L3: (an existing file name)
    path/name of file with the 3rd eigenvalue
MD: (an existing file name)
    path/name of file with the mean diffusivity
MO: (an existing file name)
    path/name of file with the mode of anisotropy
S0: (an existing file name)
    path/name of file with the raw T2 signal with no diffusion weighting
V1: (an existing file name)
    path/name of file with the 1st eigenvector
V2: (an existing file name)
    path/name of file with the 2nd eigenvector
V3: (an existing file name)
    path/name of file with the 3rd eigenvector
tensor: (an existing file name)
    path/name of file with the 4D tensor volume

```

References:: None

DistanceMap

[Link to code](#)

Wraps command **distancemap**

Use FSL's distancemap to generate a map of the distance to the nearest nonzero voxel.

Example

```
>>> import nipy.interfaces.fsl as fsl
>>> mapper = fsl.DistanceMap()
>>> mapper.inputs.in_file = "skeleton_mask.nii.gz"
>>> mapper.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to calculate distance values for
        flag: --in=%s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
distance_map: (a file name)
              distance map to write
              flag: --out=%s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
invert_input: (a boolean)
              invert input image
              flag: --invert
local_max_file: (a boolean or a file name)
                write an image of the local maxima
                flag: --localmax=%s
mask_file: (an existing file name)
            binary mask to constrain calculations
            flag: --mask=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
distance_map: (an existing file name)
              value is distance to nearest nonzero voxels
local_max_file: (a file name)
                image of local maxima
```

References:: None

FindTheBiggest

[Link to code](#)

Wraps command **find_the_biggest**

Use FSL `find_the_biggest` for performing hard segmentation on the outputs of connectivity-based thresholding in probtrack. For complete details, see the [FDT Documentation](#).

Example

```
>>> from nipy.interfaces import fsl
>>> ldir = ['seeds_to_M1.nii', 'seeds_to_M2.nii']
>>> fBig = fsl.FindTheBiggest(in_files=ldir, out_file='biggestSegmentation')
>>> fBig.cmdline
'find_the_biggest seeds_to_M1.nii seeds_to_M2.nii biggestSegmentation'
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
          a list of input volumes or a singleMatrixFile
          flag: %s, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          file with the resulting segmentation
          flag: %s, position: 2
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          output file indexed in order of input files
          flag: %s
```

References:: None

MakeDyadicVectors

[Link to code](#)

Wraps command **make_dyadic_vectors**

Create vector volume representing mean principal diffusion direction and its uncertainty (dispersion)

Inputs:

```
[Mandatory]
phi_vol: (an existing file name)
        flag: %s, position: 1
theta_vol: (an existing file name)
        flag: %s, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyype default value: {})
          Environment variables
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask: (an existing file name)
      flag: %s, position: 2
output: (a file name, nipyype default value: dyads)
        flag: %s, position: 3
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
            FSL output type
perc: (a float)
      the {perc}% angle of the output cone of uncertainty (output will be
      in degrees)
      flag: %f, position: 4
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
dispersion: (an existing file name)
dyads: (an existing file name)
```

References:: None

ProbTrackX

[Link to code](#)

Wraps command **probtrackx**

Use FSL probtrackx for tractography on bedpostx results

Examples

```
>>> from nipyype.interfaces import fsl
>>> pbx = fsl.ProbTrackX(samples_base_name='merged', mask='mask.nii', seed=
↳ 'MASK_average_thal_right.nii', mode='seedmask', xfm='trans.mat', n_
↳ samples=3, n_steps=10, force_dir=True, opd=True, os2t=True, target_masks = [
↳ 'targets_MASK1.nii', 'targets_MASK2.nii'], thsamples='merged_thsamples.nii',
↳ fsamples='merged_fsamples.nii', phsamples='merged_phsamples.nii', out_dir=
↳ '.')
>>> pbx.cmdline
'probtrackx --forcedir -m mask.nii --mode=seedmask --nsamples=3 --nsteps=10 --opd_
↳ --os2t --dir=. --samples=merged --seed=MASK_average_thal_right.nii --
↳ targetmasks=targets.txt --xfm=trans.mat'
```

Inputs:

```

[Mandatory]
fsamples: (a list of items which are an existing file name)
mask: (an existing file name)
      bet binary mask file in diffusion space
      flag: -m %s
phsamples: (a list of items which are an existing file name)
seed: (an existing file name or a list of items which are an existing
      file name or a list of items which are a list of from 3 to 3 items
      which are an integer (int or long))
      seed volume(s), or voxel(s) or freesurfer label file
      flag: --seed=%s
thsamples: (a list of items which are an existing file name)

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
avoid_mp: (an existing file name)
      reject pathways passing through locations given by this mask
      flag: --avoid=%s
c_thresh: (a float)
      curvature threshold - default=0.2
      flag: --cthr=%.3f
correct_path_distribution: (a boolean)
      correct path distribution for the length of the pathways
      flag: --pd
dist_thresh: (a float)
      discards samples shorter than this threshold (in mm - default=0)
      flag: --distthresh=%.3f
environ: (a dictionary with keys which are a bytes or None or a value
      of class 'str' and with values which are a bytes or None or a value
      of class 'str', nipyype default value: {})
      Environment variables
fibst: (an integer (int or long))
      force a starting fibre for tracking - default=1, i.e. first fibre
      orientation. Only works if randfib==0
      flag: --fibst=%d
force_dir: (a boolean, nipyype default value: True)
      use the actual directory name given - i.e. do not add + to make a
      new directory
      flag: --forcedir
ignore_exception: (a boolean, nipyype default value: False)
      Print an error message instead of throwing an exception in case the
      interface fails to run
inv_xfm: (a file name)
      transformation matrix taking DTI space to seed space (compulsory
      when using a warp_field for seeds_to_dti)
      flag: --invxfm=%s
loop_check: (a boolean)
      perform loop_checks on paths - slower, but allows lower curvature
      threshold
      flag: --loopcheck
mask2: (an existing file name)
      second bet binary mask (in diffusion space) in twomask_symm mode
      flag: --mask2=%s

```

```
mesh: (an existing file name)
    Freesurfer-type surface descriptor (in ascii format)
    flag: --mesh=%s
mod_euler: (a boolean)
    use modified euler streamlining
    flag: --modeuler
mode: ('simple' or 'two_mask_symm' or 'seedmask')
    options: simple (single seed voxel), seedmask (mask of seed voxels),
    twomask_symm (two bet binary masks)
    flag: --mode=%s
n_samples: (an integer (int or long), nipy default value: 5000)
    number of samples - default=5000
    flag: --nsamples=%d
n_steps: (an integer (int or long))
    number of steps per sample - default=2000
    flag: --nsteps=%d
network: (a boolean)
    activate network mode - only keep paths going through at least one
    seed mask (required if multiple seed masks)
    flag: --network
opd: (a boolean, nipy default value: True)
    outputs path distributions
    flag: --opd
os2t: (a boolean)
    Outputs seeds to targets
    flag: --os2t
out_dir: (an existing directory name)
    directory to put the final volumes in
    flag: --dir=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
rand_fib: (0 or 1 or 2 or 3)
    options: 0 - default, 1 - to randomly sample initial fibres (with f
    > fibthresh), 2 - to sample in proportion fibres (with f>fibthresh)
    to f, 3 - to sample ALL populations at random (even if f<fibthresh)
    flag: --randfib=%d
random_seed: (a boolean)
    random seed
    flag: --rseed
s2tastext: (a boolean)
    output seed-to-target counts as a text file (useful when seeding
    from a mesh)
    flag: --s2tastext
sample_random_points: (a boolean)
    sample random points within seed voxels
    flag: --sampvox
samples_base_name: (a unicode string, nipy default value: merged)
    the rootname/base_name for samples files
    flag: --samples=%s
seed_ref: (an existing file name)
    reference vol to define seed space in simple mode - diffusion space
    assumed if absent
    flag: --seedref=%s
step_length: (a float)
    step_length in mm - default=0.5
    flag: --steplength=%.3f
stop_mask: (an existing file name)
```

```

        stop tracking at locations given by this mask file
        flag: --stop=%s
target_masks: (a list of items which are a file name)
        list of target masks - required for seeds_to_targets classification
        flag: --targetmasks=%s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
use_anisotropy: (a boolean)
        use anisotropy to constrain tracking
        flag: --usef
verbose: (0 or 1 or 2)
        Verbose level, [0-2]. Level 2 is required to output particle files.
        flag: --verbose=%d
waypoints: (an existing file name)
        waypoint mask or ascii list of waypoint masks - only keep paths
        going through ALL the masks
        flag: --waypoints=%s
xfm: (an existing file name)
        transformation matrix taking seed space to DTI space (either FLIRT
        matrix or FNIRT warp_field) - default is identity
        flag: --xfm=%s

```

Outputs:

```

fdt_paths: (a list of items which are an existing file name)
        path/name of a 3D image file containing the output connectivity
        distribution to the seed mask
log: (an existing file name)
        path/name of a text record of the command that was run
particle_files: (a list of items which are an existing file name)
        Files describing all of the tract samples. Generated only if verbose
        is set to 2
targets: (a list of items which are an existing file name)
        a list with all generated seeds_to_target files
way_total: (an existing file name)
        path/name of a text file containing a single number corresponding to
        the total number of generated tracts that have not been rejected by
        inclusion/exclusion mask criteria

```

References:: None

ProbTrackX2[Link to code](#)Wraps command **probtrackx2**

Use FSL probtrackx2 for tractography on bedpostx results

Examples

```

>>> from nipy.interfaces import fsl
>>> pbx2 = fsl.ProbTrackX2()
>>> pbx2.inputs.seed = 'seed_source.nii.gz'
>>> pbx2.inputs.thsamples = 'merged_thlsamples.nii.gz'
>>> pbx2.inputs.fsamples = 'merged_flsamples.nii.gz'
>>> pbx2.inputs.phsamples = 'merged_phlsamples.nii.gz'
>>> pbx2.inputs.mask = 'nodif_brain_mask.nii.gz'

```

```

>>> pbx2.inputs.out_dir = '.'
>>> pbx2.inputs.n_samples = 3
>>> pbx2.inputs.n_steps = 10
>>> pbx2.cmdline
'probtrackx2 --forcedir -m nodif_brain_mask.nii.gz --nsamples=3 --nsteps=10 --opd_
↪--dir=. --samples=merged --seed=seed_source.nii.gz'

```

Inputs:

```

[Mandatory]
fsamples: (a list of items which are an existing file name)
mask: (an existing file name)
      bet binary mask file in diffusion space
      flag: -m %s
phsamples: (a list of items which are an existing file name)
seed: (an existing file name or a list of items which are an existing
      file name or a list of items which are a list of from 3 to 3 items
      which are an integer (int or long))
      seed volume(s), or voxel(s) or freesurfer label file
      flag: --seed=%s
thsamples: (a list of items which are an existing file name)

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
avoid_mp: (an existing file name)
      reject pathways passing through locations given by this mask
      flag: --avoid=%s
c_thresh: (a float)
      curvature threshold - default=0.2
      flag: --cthr=%.3f
colmask4: (an existing file name)
      Mask for columns of matrix4 (default=seed mask)
      flag: --colmask4=%s
correct_path_distribution: (a boolean)
      correct path distribution for the length of the pathways
      flag: --pd
dist_thresh: (a float)
      discards samples shorter than this threshold (in mm - default=0)
      flag: --distthresh=%.3f
distthresh1: (a float)
      Discards samples (in matrix1) shorter than this threshold (in mm -
      default=0)
      flag: --distthresh1=%.3f
distthresh3: (a float)
      Discards samples (in matrix3) shorter than this threshold (in mm -
      default=0)
      flag: --distthresh3=%.3f
environ: (a dictionary with keys which are a bytes or None or a value
      of class 'str' and with values which are a bytes or None or a value
      of class 'str', nipy default value: {})
      Environment variables
fibst: (an integer (int or long))
      force a starting fibre for tracking - default=1, i.e. first fibre
      orientation. Only works if randfib==0
      flag: --fibst=%d
fopd: (an existing file name)

```



```

    Other mask for binning tract distribution
    flag: --fopd=%s
force_dir: (a boolean, nipy default value: True)
    use the actual directory name given - i.e. do not add + to make a
    new directory
    flag: --forcedir
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inv_xfm: (a file name)
    transformation matrix taking DTI space to seed space (compulsory
    when using a warp_field for seeds_to_dti)
    flag: --invxfm=%s
loop_check: (a boolean)
    perform loop_checks on paths - slower, but allows lower curvature
    threshold
    flag: --loopcheck
lrtarget3: (an existing file name)
    Column-space mask used for NxN connectivity matrix
    flag: --lrtarget3=%s
meshspace: ('caret' or 'freesurfer' or 'first' or 'vox')
    Mesh reference space - either "caret" (default) or "freesurfer" or
    "first" or "vox"
    flag: --meshspace=%s
mod_euler: (a boolean)
    use modified euler streamlining
    flag: --modeuler
n_samples: (an integer (int or long), nipy default value: 5000)
    number of samples - default=5000
    flag: --nsamples=%d
n_steps: (an integer (int or long))
    number of steps per sample - default=2000
    flag: --nsteps=%d
network: (a boolean)
    activate network mode - only keep paths going through at least one
    seed mask (required if multiple seed masks)
    flag: --network
omatrix1: (a boolean)
    Output matrix1 - SeedToSeed Connectivity
    flag: --omatrix1
omatrix2: (a boolean)
    Output matrix2 - SeedToLowResMask
    flag: --omatrix2
    requires: target2
omatrix3: (a boolean)
    Output matrix3 (NxN connectivity matrix)
    flag: --omatrix3
    requires: target3, lrtarget3
omatrix4: (a boolean)
    Output matrix4 - DtiMaskToSeed (special Oxford Sparse Format)
    flag: --omatrix4
onewaycondition: (a boolean)
    Apply waypoint conditions to each half tract separately
    flag: --onewaycondition
opd: (a boolean, nipy default value: True)
    outputs path distributions
    flag: --opd
os2t: (a boolean)

```

```

        Outputs seeds to targets
        flag: --os2t
out_dir: (an existing directory name)
        directory to put the final volumes in
        flag: --dir=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
        FSL output type
rand_fib: (0 or 1 or 2 or 3)
        options: 0 - default, 1 - to randomly sample initial fibres (with f
        > fibthresh), 2 - to sample in proportion fibres (with f>fibthresh)
        to f, 3 - to sample ALL populations at random (even if f<fibthresh)
        flag: --randfib=%d
random_seed: (a boolean)
        random seed
        flag: --rseed
s2tastext: (a boolean)
        output seed-to-target counts as a text file (useful when seeding
        from a mesh)
        flag: --s2tastext
sample_random_points: (a boolean)
        sample random points within seed voxels
        flag: --sampvox
samples_base_name: (a unicode string, nipy default value: merged)
        the rootname/base_name for samples files
        flag: --samples=%s
seed_ref: (an existing file name)
        reference vol to define seed space in simple mode - diffusion space
        assumed if absent
        flag: --seedref=%s
simple: (a boolean)
        rack from a list of voxels (seed must be a ASCII list of
        coordinates)
        flag: --simple
step_length: (a float)
        step_length in mm - default=0.5
        flag: --steplength=%.3f
stop_mask: (an existing file name)
        stop tracking at locations given by this mask file
        flag: --stop=%s
target2: (an existing file name)
        Low resolution binary brain mask for storing connectivity
        distribution in matrix2 mode
        flag: --target2=%s
target3: (an existing file name)
        Mask used for NxN connectivity matrix (or NxN if lrtarget3 is set)
        flag: --target3=%s
target4: (an existing file name)
        Brain mask in DTI space
        flag: --target4=%s
target_masks: (a list of items which are a file name)
        list of target masks - required for seeds_to_targets classification
        flag: --targetmasks=%s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
use_anisotropy: (a boolean)

```

```

        use anisotropy to constrain tracking
        flag: --usef
verbose: (0 or 1 or 2)
        Verbose level, [0-2]. Level 2 is required to output particle files.
        flag: --verbose=%d
waycond: ('OR' or 'AND')
        Waypoint condition. Either "AND" (default) or "OR"
        flag: --waycond=%s
wayorder: (a boolean)
        Reject streamlines that do not hit waypoints in given order. Only
        valid if waycond=AND
        flag: --wayorder
waypoints: (an existing file name)
        waypoint mask or ascii list of waypoint masks - only keep paths
        going through ALL the masks
        flag: --waypoints=%s
xfm: (an existing file name)
        transformation matrix taking seed space to DTI space (either FLIRT
        matrix or FNIRT warp_field) - default is identity
        flag: --xfm=%s

```

Outputs:

```

fdt_paths: (a list of items which are an existing file name)
        path/name of a 3D image file containing the output connectivity
        distribution to the seed mask
log: (an existing file name)
        path/name of a text record of the command that was run
lookup_tractspace: (an existing file name)
        lookup_tractspace generated by --omatrix2 option
matrix1_dot: (an existing file name)
        Output matrix1.dot - SeedToSeed Connectivity
matrix2_dot: (an existing file name)
        Output matrix2.dot - SeedToLowResMask
matrix3_dot: (an existing file name)
        Output matrix3 - NxN connectivity matrix
network_matrix: (an existing file name)
        the network matrix generated by --omatrix1 option
particle_files: (a list of items which are an existing file name)
        Files describing all of the tract samples. Generated only if verbose
        is set to 2
targets: (a list of items which are an existing file name)
        a list with all generated seeds_to_target files
way_total: (an existing file name)
        path/name of a text file containing a single number corresponding to
        the total number of generated tracts that have not been rejected by
        inclusion/exclusion mask criteria

```

References:: None

ProjThresh[Link to code](#)Wraps command **proj_thresh**

Use FSL **proj_thresh** for thresholding some outputs of probtrack For complete details, see the FDT Documenta**tion** <http://www.fmrib.ox.ac.uk/fsl/fdt/fdt_thresh.html>

Example

```
>>> from nipyype.interfaces import fsl
>>> ldir = ['seeds_to_M1.nii', 'seeds_to_M2.nii']
>>> pThresh = fsl.ProjThresh(in_files=ldir, threshold=3)
>>> pThresh.cmdline
'proj_thresh seeds_to_M1.nii seeds_to_M2.nii 3'
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
          a list of input volumes
          flag: %s, position: 0
threshold: (an integer (int or long))
           threshold indicating minimum number of seed voxels entering this
           mask region
           flag: %d, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyype default value: {})
          Environment variables
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_files: (a list of items which are an existing file name)
            path/name of output volume after thresholding
```

References:: None

TractSkeleton

[Link to code](#)

Wraps command **tbss_skeleton**

Use FSL's `tbss_skeleton` to skeletonise an FA image or project arbitrary values onto a skeleton.

There are two ways to use this interface. To create a skeleton from an FA image, just supply the `in_file` and set `skeleton_file` to `True` (or specify a skeleton filename). To project values onto a skeleton, you must set `project_data` to `True`, and then also supply values for `threshold`, `distance_map`, and `data_file`. The `search_mask_file` and `use_cingulum_mask` inputs are also used in data projection, but `use_cingulum_mask` is set to `True` by default. This mask controls where the projection algorithm searches within a circular space around a tract, rather than in a single perpendicular direction.

Example

```
>>> import nipy.interfaces.fsl as fsl
>>> skeleton = fsl.TractSkeleton()
>>> skeleton.inputs.in_file = "all_FA.nii.gz"
>>> skeleton.inputs.skeleton_file = True
>>> skeleton.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input image (typcially mean FA volume)
         flag: -i %s

[Optional]
alt_data_file: (an existing file name)
               4D non-FA data to project onto skeleton
               flag: -a %s
alt_skeleton: (an existing file name)
              alternate skeleton to use
              flag: -s %s
args: (a unicode string)
      Additional parameters to the command
      flag: %s
data_file: (an existing file name)
           4D data to project onto skeleton (usually FA)
distance_map: (an existing file name)
              distance map image
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
project_data: (a boolean)
              project data onto skeleton
              flag: -p %.3f %s %s %s %s
              requires: threshold, distance_map, data_file
projected_data: (a file name)
                input data projected onto skeleton
search_mask_file: (an existing file name)
                  mask in which to use alternate search rule
                  mutually_exclusive: use_cingulum_mask
skeleton_file: (a boolean or a file name)
                write out skeleton image
                flag: -o %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
threshold: (a float)
            skeleton threshold value
use_cingulum_mask: (a boolean, nipy default value: True)
                   perform alternate search using built-in cingulum mask
```

```
mutually_exclusive: search_mask_file
```

Outputs:

```
projected_data: (a file name)
    input data projected onto skeleton
skeleton_file: (a file name)
    tract skeleton image
```

References:: None

VecReg

[Link to code](#)

Wraps command **vecreg**

Use FSL vecreg for registering vector data For complete details, see the FDT Documentation <http://www.fmrib.ox.ac.uk/fsl/fdt/fdt_vecreg.html>

Example

```
>>> from nipyne.interfaces import fsl
>>> vreg = fsl.VecReg(in_file='diffusion.nii',                affine_mat='trans.
↪mat',                ref_vol='mni.nii',                    out_file='diffusion_
↪vreg.nii')
>>> vreg.cmdline
'vecreg -t trans.mat -i diffusion.nii -o diffusion_vreg.nii -r mni.nii'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    filename for input vector or tensor field
    flag: -i %s
ref_vol: (an existing file name)
    filename for reference (target) volume
    flag: -r %s

[Optional]
affine_mat: (an existing file name)
    filename for affine transformation matrix
    flag: -t %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interpolation: ('nearestneighbour' or 'trilinear' or 'sinc' or
    'spline')
    interpolation method : nearestneighbour, trilinear (default), sinc
    or spline
    flag: --interp=%s
mask: (an existing file name)
    brain mask in input space
```

```

    flag: -m %s
out_file: (a file name)
    filename for output registered vector or tensor field
    flag: -o %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
ref_mask: (an existing file name)
    brain mask in output space (useful for speed up of nonlinear reg)
    flag: --refmask=%s
rotation_mat: (an existing file name)
    filename for secondary affine matrix if set, this will be used for
    the rotation of the vector/tensor field
    flag: --rotmat=%s
rotation_warp: (an existing file name)
    filename for secondary warp field if set, this will be used for the
    rotation of the vector/tensor field
    flag: --rotwarp=%s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
warp_field: (an existing file name)
    filename for 4D warp field for nonlinear registration
    flag: -w %s

```

Outputs:

```

out_file: (an existing file name)
    path/name of filename for the registered vector or tensor field

```

References:: None

XFibres5[Link to code](#)Wraps command **xfibres**

Perform model parameters estimation for local (voxelwise) diffusion parameters

Inputs:

```

[Mandatory]
bvals: (an existing file name)
    b values file
    flag: --bvals=%s
bvecs: (an existing file name)
    b vectors file
    flag: --bvecs=%s
dwi: (an existing file name)
    diffusion weighted image data file
    flag: --data=%s
mask: (an existing file name)
    brain binary mask file (i.e. from BET)
    flag: --mask=%s
n_fibres: (a long integer >= 1, nipy default value: 1)
    Maximum number of fibres to fit in each voxel
    flag: --nfibres=%d

[Optional]
all_ard: (a boolean)

```

```
    Turn ARD on on all fibres
    flag: --allard
    mutually_exclusive: no_ard, all_ard
args: (a unicode string)
    Additional parameters to the command
    flag: %s
burn_in: (a long integer >= 0)
    Total num of jumps at start of MCMC to be discarded
    flag: --burnin=%d
burn_in_no_ard: (a long integer >= 0)
    num of burnin jumps before the ard is imposed
    flag: --burninnoard=%d
cnonlinear: (a boolean)
    Initialise with constrained nonlinear fitting
    flag: --cnonlinear
    mutually_exclusive: no_spat, non_linear, cnonlinear
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
f0_ard: (a boolean)
    Noise floor model: add to the model an unattenuated signal
    compartment f0
    flag: --f0 --ardf0
    mutually_exclusive: f0_noard, f0_ard, all_ard
f0_noard: (a boolean)
    Noise floor model: add to the model an unattenuated signal
    compartment f0
    flag: --f0
    mutually_exclusive: f0_noard, f0_ard
force_dir: (a boolean, nipyne default value: True)
    use the actual directory name given (do not add + to make a new
    directory)
    flag: --forcedir
fudge: (an integer (int or long))
    ARD fudge factor
    flag: --fudge=%d
gradnonlin: (an existing file name)
    gradient file corresponding to slice
    flag: --gradnonlin=%s
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
logdir: (a directory name, nipyne default value: .)
    flag: --logdir=%s
model: (1 or 2 or 3)
    use monoexponential (1, default, required for single-shell) or
    multiexponential (2, multi-shell) model
    flag: --model=%d
n_jumps: (an integer (int or long))
    Num of jumps to be made by MCMC
    flag: --njumps=%d
no_ard: (a boolean)
    Turn ARD off on all fibres
    flag: --noard
    mutually_exclusive: no_ard, all_ard
no_spat: (a boolean)
    Initialise with tensor, not spatially
```



```

        flag: --nospat
        mutually_exclusive: no_spat, non_linear, cilinear
non_linear: (a boolean)
    Initialise with nonlinear fitting
    flag: --nonlinear
    mutually_exclusive: no_spat, non_linear, cilinear
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
    FSL output type
rician: (a boolean)
    use Rician noise modeling
    flag: --rician
sample_every: (a long integer >= 0)
    Num of jumps for each sample (MCMC)
    flag: --sampleevery=%d
seed: (an integer (int or long))
    seed for pseudo random number generator
    flag: --seed=%d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
update_proposal_every: (a long integer >= 1)
    Num of jumps for each update to the proposal density std (MCMC)
    flag: --updateproposalevery=%d

```

Outputs:

```

dyads: (a list of items which are an existing file name)
    Mean of PDD distribution in vector form.
fsamples: (a list of items which are an existing file name)
    Samples from the distribution on f anisotropy
mean_S0samples: (an existing file name)
    Mean of distribution on T2wbaseline signal intensity S0
mean_dsamples: (an existing file name)
    Mean of distribution on diffusivity d
mean_fsamples: (a list of items which are an existing file name)
    Mean of distribution on f anisotropy
mean_tausamples: (an existing file name)
    Mean of distribution on tau samples (only with rician noise)
phsamples: (a list of items which are an existing file name)
    phi samples, per fiber
thsamples: (a list of items which are an existing file name)
    theta samples, per fiber

```

References:: None

interfaces.fsl.epi**ApplyTOPUP**[Link to code](#)Wraps command **applytopup**Interface for FSL topup, a tool for estimating and correcting susceptibility induced distortions. [General reference](#) and [use example](#).

Examples

```
>>> from nipy.interfaces.fsl import ApplyTOPUP
>>> applytopup = ApplyTOPUP()
>>> applytopup.inputs.in_files = ["epi.nii", "epi_rev.nii"]
>>> applytopup.inputs.encoding_file = "topup_encoding.txt"
>>> applytopup.inputs.in_topup_fieldcoef = "topup_fieldcoef.nii.gz"
>>> applytopup.inputs.in_topup_movpar = "topup_movpar.txt"
>>> applytopup.inputs.output_type = "NIFTI_GZ"
>>> applytopup.cmdline
'applytopup --datain=topup_encoding.txt --imain=epi.nii,epi_rev.nii --inindex=1,2,
↳--topup=topup --out=epi_corrected.nii.gz'
>>> res = applytopup.run()
```

Inputs:

```
[Mandatory]
encoding_file: (an existing file name)
    name of text file with PE directions/times
    flag: --datain=%s
in_files: (a list of items which are an existing file name)
    name of file with images
    flag: --imain=%s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
datatype: ('char' or 'short' or 'int' or 'float' or 'double')
    force output data type
    flag: -d=%s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_index: (a list of items which are an integer (int or long))
    comma separated list of indices corresponding to --datain
    flag: --inindex=%s
in_topup_fieldcoef: (an existing file name)
    topup file containing the field coefficients
    flag: --topup=%s
    requires: in_topup_movpar
in_topup_movpar: (an existing file name)
    topup movpar.txt file
    requires: in_topup_fieldcoef
interp: ('trilinear' or 'spline')
    interpolation method
    flag: --interp=%s
method: ('jac' or 'lsr')
    use jacobian modulation (jac) or least-squares resampling (lsr)
    flag: --method=%s
out_corrected: (a file name)
    output (warped) image
    flag: --out=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
```

```
FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_corrected: (an existing file name)
               name of 4D image file with unwarped images
```

References:: None

EPIDeWarp[Link to code](#)Wraps command **epidewarp.fsl**Wraps the unwarping script [epidewarp.fsl](#).

Warning: deprecated in FSL, please use `nipy.workflows.dmri.preprocess.epi.sdc_fmb()` instead.

Examples

```
>>> from nipy.interfaces.fsl import EPIDeWarp
>>> dewarp = EPIDeWarp()
>>> dewarp.inputs.epi_file = "functional.nii"
>>> dewarp.inputs.mag_file = "magnitude.nii"
>>> dewarp.inputs.dph_file = "phase.nii"
>>> dewarp.inputs.output_type = "NIFTI_GZ"
>>> dewarp.cmdline
'epidewarp.fsl --mag magnitude.nii --dph phase.nii --epi functional.nii --esp 0.
↪58 --exfdw ../exfdw.nii.gz --nocleanup --sigma 2 --tediff 2.46 --tmpdir ../
↪temp --vsm ../vsm.nii.gz'
>>> res = dewarp.run()
```

Inputs:

```
[Mandatory]
dph_file: (an existing file name)
          Phase file assumed to be scaled from 0 to 4095
          flag: --dph %s
mag_file: (an existing file name)
          Magnitude file
          flag: --mag %s, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
cleanup: (a boolean)
         cleanup
         flag: --cleanup
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
```

```

epi_file: (an existing file name)
    EPI volume to unwarp
    flag: --epi %s
epidw: (a string)
    dewarped epi volume
    flag: --epidw %s
esp: (a float, nipy default value: 0.58)
    EPI echo spacing
    flag: --esp %s
exf_file: (an existing file name)
    example func volume (or use epi)
    flag: --exf %s
exfdw: (a string)
    dewarped example func volume
    flag: --exfdw %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
nocleanup: (a boolean, nipy default value: True)
    no cleanup
    flag: --nocleanup
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
sigma: (an integer (int or long), nipy default value: 2)
    2D spatial gaussing smoothing stdev (default = 2mm)
    flag: --sigma %s
tediff: (a float, nipy default value: 2.46)
    difference in B0 field map TEs
    flag: --tediff %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tmpdir: (a string)
    tmpdir
    flag: --tmpdir %s
vsm: (a string)
    voxel shift map
    flag: --vsm %s

```

Outputs:

```

exf_mask: (a file name)
    Mask from example functional volume
exfdw: (a file name)
    dewarped functional volume example
unwarped_file: (a file name)
    unwarped epi file
vsm_file: (a file name)
    voxel shift map

```

References:: None

Eddy[Link to code](#)Wraps command **eddy_openmp**Interface for FSL eddy, a tool for estimating and correcting eddy currents induced distortions. [User guide](#) and

more info regarding acqp file.

Examples

```
>>> from nipyre.interfaces.fsl import Eddy
>>> eddy = Eddy()
>>> eddy.inputs.in_file = 'epi.nii'
>>> eddy.inputs.in_mask = 'epi_mask.nii'
>>> eddy.inputs.in_index = 'epi_index.txt'
>>> eddy.inputs.in_acqp = 'epi_acqp.txt'
>>> eddy.inputs.in_bvec = 'bvecs.scheme'
>>> eddy.inputs.in_bval = 'bvals.scheme'
>>> eddy.inputs.use_cuda = True
>>> eddy.cmdline
'eddy_cuda --acqp=epi_acqp.txt --bvals=bvals.scheme --bvecs=bvecs.scheme --
↳ imain=epi.nii --index=epi_index.txt --mask=epi_mask.nii --out=.../eddy_corrected
↳ '
>>> eddy.inputs.use_cuda = False
>>> eddy.cmdline
'eddy_openmp --acqp=epi_acqp.txt --bvals=bvals.scheme --bvecs=bvecs.scheme --
↳ imain=epi.nii --index=epi_index.txt --mask=epi_mask.nii --out=.../eddy_corrected
↳ '
>>> res = eddy.run()
```

Inputs:

```
[Mandatory]
in_acqp: (an existing file name)
    File containing acquisition parameters
    flag: --acqp=%s
in_bval: (an existing file name)
    File containing the b-values for all volumes in --imain
    flag: --bvals=%s
in_bvec: (an existing file name)
    File containing the b-vectors for all volumes in --imain
    flag: --bvecs=%s
in_file: (an existing file name)
    File containing all the images to estimate distortions for
    flag: --imain=%s
in_index: (an existing file name)
    File containing indices for all volumes in --imain into --acqp and
    --topup
    flag: --index=%s
in_mask: (an existing file name)
    Mask to indicate brain
    flag: --mask=%s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dont_peas: (a boolean)
    Do NOT perform a post-eddy alignment of shells
    flag: --dont_peas
dont_sep_offs_move: (a boolean)
    Do NOT attempt to separate field offset from subject movement
    flag: --dont_sep_offs_move
environ: (a dictionary with keys which are a bytes or None or a value
```

```

        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
    Environment variables
fep: (a boolean)
    Fill empty planes in x- or y-directions
    flag: --fep
field: (a unicode string)
    NonTOPUP fieldmap scaled in Hz - filename has to be provided without
    an extension. TOPUP is strongly recommended
    flag: --field=%s
field_mat: (an existing file name)
    Matrix that specifies the relative locations of the field specified
    by --field and first volume in file --imain
    flag: --field_mat=%s
flm: ('linear' or 'quadratic' or 'cubic')
    First level EC model
    flag: --flm=%s
fudge_factor: (a float)
    Fudge factor for hyperparameter error variance
    flag: --ff=%s
fwhm: (a float)
    FWHM for conditioning filter when estimating the parameters
    flag: --fwhm=%s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_topup_fieldcoef: (an existing file name)
    topup file containing the field coefficients
    flag: --topup=%s
    requires: in_topup_movpar
in_topup_movpar: (an existing file name)
    topup movpar.txt file
    requires: in_topup_fieldcoef
interp: ('spline' or 'trilinear')
    Interpolation model for estimation step
    flag: --interp=%s
is_shelled: (a boolean)
    Override internal check to ensure that data are acquired on a set of
    b-value shells
    flag: --data_is_shelled
method: ('jac' or 'lsr')
    Final resampling method (jacobian/least squares)
    flag: --resamp=%s
niter: (an integer (int or long))
    Number of iterations
    flag: --niter=%s
num_threads: (an integer (int or long), nipy default value: 1)
    Number of openmp threads to use
nvoxhp: (an integer (int or long))
    # of voxels used to estimate the hyperparameters
    flag: --nvoxhp=%s
out_base: (a unicode string, nipy default value: eddy_corrected)
    basename for output (warped) image
    flag: --out=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
repol: (a boolean)

```

```

    Detect and replace outlier slices
    flag: --repol
session: (an existing file name)
    File containing session indices for all volumes in --imain
    flag: --session=%s
slm: ('none' or 'linear' or 'quadratic')
    Second level EC model
    flag: --slm=%s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
use_cuda: (a boolean)
    Run eddy using cuda gpu

```

Outputs:

```

out_corrected: (an existing file name)
    4D image file containing all the corrected volumes
out_movement_rms: (an existing file name)
    Summary of the "total movement" in each volume
out_outlier_report: (an existing file name)
    Text-file with a plain language report on what outlier slices eddy
    has found
out_parameter: (an existing file name)
    text file with parameters definining the field andmovement for each
    scan
out_restricted_movement_rms: (an existing file name)
    Summary of the "total movement" in each volume disregarding
    translation in the PE direction
out_rotated_bvecs: (an existing file name)
    File containing rotated b-values for all volumes
out_shell_alignment_parameters: (an existing file name)
    File containing rigid body movement parameters between the different
    shells as estimated by a post-hoc mutual information based
    registration

```

References:: None

EddyCorrect[Link to code](#)Wraps command **eddy_correct**

Warning: Deprecated in FSL. Please use `nipy.interfaces.fsl.epi.Eddy` instead

Example

```

>>> from nipy.interfaces.fsl import EddyCorrect
>>> eddyc = EddyCorrect(in_file='diffusion.nii',
...                     out_file="diffusion_edc.nii", ref_num=0)
>>> eddyc.cmdline
'eddy_correct diffusion.nii diffusion_edc.nii 0'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        4D input file
        flag: %s, position: 0
ref_num: (an integer (int or long), nipy default value: 0)
        reference number
        flag: %d, position: 2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
out_file: (a file name)
        4D output file
        flag: %s, position: 1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_PAIR_GZ')
        FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

eddy_corrected: (an existing file name)
                path/name of 4D eddy corrected output file

```

References:: None

EpiReg[Link to code](#)Wraps command **epi_reg**

Runs FSL epi_reg script for simultaneous coregistration and fieldmap unwarping.

Examples

```

>>> from nipy.interfaces.fsl import EpiReg
>>> epiereg = EpiReg()
>>> epiereg.inputs.epi='epi.nii'
>>> epiereg.inputs.t1_head='T1.nii'
>>> epiereg.inputs.t1_brain='T1_brain.nii'
>>> epiereg.inputs.out_base='epi2struct'
>>> epiereg.inputs.fmap='fieldmap_phase_fslprepared.nii'
>>> epiereg.inputs.fmapmag='fieldmap_mag.nii'
>>> epiereg.inputs.fmapmagbrain='fieldmap_mag_brain.nii'
>>> epiereg.inputs.echospacing=0.00067
>>> epiereg.inputs.pedir='y'
>>> epiereg.cmdline

```



```
'epi_reg --echospacing=0.000670 --fmap=fieldmap_phase_fslprepared.nii --
→fmapmag=fieldmap_mag.nii --fmapmagbrain=fieldmap_mag_brain.nii --noclean --
→pedir=y --epi=epi.nii --t1=T1.nii --t1brain=T1_brain.nii --out=epi2struct'
>>> epireg.run()
```

Inputs:

```
[Mandatory]
epi: (an existing file name)
    EPI image
    flag: --epi=%s, position: -4
t1_brain: (an existing file name)
    brain extracted T1 image
    flag: --t1brain=%s, position: -2
t1_head: (an existing file name)
    wholehead T1 image
    flag: --t1=%s, position: -3

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
echospacing: (a float)
    Effective EPI echo spacing (sometimes called dwell time) - in
    seconds
    flag: --echospacing=%f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fmap: (an existing file name)
    fieldmap image (in rad/s)
    flag: --fmap=%s
fmapmag: (an existing file name)
    fieldmap magnitude image - wholehead
    flag: --fmapmag=%s
fmapmagbrain: (an existing file name)
    fieldmap magnitude image - brain extracted
    flag: --fmapmagbrain=%s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
no_clean: (a boolean, nipy default value: True)
    do not clean up intermediate files
    flag: --noclean
no_fmapreg: (a boolean)
    do not perform registration of fmap to T1 (use if fmap already
    registered)
    flag: --nofmapreg
out_base: (a string, nipy default value: epi2struct)
    output base name
    flag: --out=%s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
pedir: ('x' or 'y' or 'z' or '-x' or '-y' or '-z')
    phase encoding direction, dir = x/y/z/-x/-y/-z
    flag: --pedir=%s
```

```

terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
weight_image: (an existing file name)
    weighting image (in T1 space)
    flag: --weight=%s
wmseg: (an existing file name)
    white matter segmentation of T1 image, has to be named like the
    tlbrain and end on _wmseg
    flag: --wmseg=%s

```

Outputs:

```

epi2str_inv: (an existing file name)
    rigid structural-to-epi transform
epi2str_mat: (an existing file name)
    rigid epi-to-structural transform
fmap2epi_mat: (an existing file name)
    rigid fieldmap-to-epi transform
fmap2str_mat: (an existing file name)
    rigid fieldmap-to-structural transform
fmap_epi: (an existing file name)
    fieldmap in epi space
fmap_str: (an existing file name)
    fieldmap in structural space
fmapmag_str: (an existing file name)
    fieldmap magnitude image in structural space
fullwarp: (an existing file name)
    warpfield to unwarp epi and transform into structural space
out_1vol: (an existing file name)
    unwarpd and coregistered single volume
out_file: (an existing file name)
    unwarpd and coregistered epi input
shiftmap: (an existing file name)
    shiftmap in epi space
wmedge: (an existing file name)
    white matter edges for visualization
wmseg: (an existing file name)
    white matter segmentation used in flirt bbr

```

References:: None

PrepareFieldmap[Link to code](#)Wraps command **fsl_prepare_fieldmap**

Interface for the fsl_prepare_fieldmap script (FSL 5.0)

Prepares a fieldmap suitable for FEAT from SIEMENS data - saves output in rad/s format (e.g. ``fsl_prepare_fieldmap SIEMENS images_3_gre_field_mapping images_4_gre_field_mapping fmap_rads 2.65``).

Examples

```

>>> from nipy.interfaces.fsl import PrepareFieldmap
>>> prepare = PrepareFieldmap()
>>> prepare.inputs.in_phase = "phase.nii"

```

```
>>> prepare.inputs.in_magnitude = "magnitude.nii"
>>> prepare.inputs.output_type = "NIFTI_GZ"
>>> prepare.cmdline
'fsl_prepare_fieldmap SIEMENS phase.nii magnitude.nii ../phase_fslprepared.nii.
↳gz 2.460000'
>>> res = prepare.run()
```

Inputs:

```
[Mandatory]
delta_TE: (a float, nipy default value: 2.46)
    echo time difference of the fieldmap sequence in ms. (usually 2.46ms
    in Siemens)
    flag: %f, position: -2
in_magnitude: (an existing file name)
    Magnitude difference map, brain extracted
    flag: %s, position: 3
in_phase: (an existing file name)
    Phase difference map, in SIEMENS format range from 0-4096 or 0-8192)
    flag: %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
nocheck: (a boolean, nipy default value: False)
    do not perform sanity checks for image size/range/dimensions
    flag: --nocheck, position: -1
out_fieldmap: (a file name)
    output name for prepared fieldmap
    flag: %s, position: 4
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
scanner: (a string, nipy default value: SIEMENS)
    must be SIEMENS
    flag: %s, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_fieldmap: (an existing file name)
    output name for prepared fieldmap
```

References: None

SigLoss[Link to code](#)

Wraps command **sigloss**

Estimates signal loss from a field map (in rad/s)

Examples

```
>>> from nipy.interfaces.fsl import SigLoss
>>> sigloss = SigLoss()
>>> sigloss.inputs.in_file = "phase.nii"
>>> sigloss.inputs.echo_time = 0.03
>>> sigloss.inputs.output_type = "NIFTI_GZ"
>>> sigloss.cmdline
'sigloss --te=0.030000 -i phase.nii -s ../phase_sigloss.nii.gz'
>>> res = sigloss.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        b0 fieldmap file
        flag: -i %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
echo_time: (a float)
           echo time in seconds
           flag: --te=%f
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
mask_file: (an existing file name)
           brain mask file
           flag: -m %s
out_file: (a file name)
          output signal loss estimate file
          flag: -s %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
slice_direction: ('x' or 'y' or 'z')
                 slicing direction
                 flag: -d %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
                 (default), `allatonce` - waits till command is finished to display
                 output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          signal loss estimate file
```

References:: None

TOPUP

[Link to code](#)

Wraps command **topup**

Interface for FSL **topup**, a tool for estimating and correcting susceptibility induced distortions. See FSL documentation for [reference](#), [usage examples](#), and [exemplary config files](#).

Examples

```
>>> from nipyype.interfaces.fsl import TOPUP
>>> topup = TOPUP()
>>> topup.inputs.in_file = "b0_b0rev.nii"
>>> topup.inputs.encoding_file = "topup_encoding.txt"
>>> topup.inputs.output_type = "NIFTI_GZ"
>>> topup.cmdline
'topup --config=b02b0.cnf --datain=topup_encoding.txt --imain=b0_b0rev.nii --
→out=b0_b0rev_base --iout=b0_b0rev_corrected.nii.gz --fout=b0_b0rev_field.nii.gz
→--jacout=jac --logout=b0_b0rev_topup.log --dfout=warpfield'
>>> res = topup.run()
```

Inputs:

```
[Mandatory]
encoding_direction: (a list of items which are 'y' or 'x' or 'z' or
                    'x-' or 'y-' or 'z-')
                    encoding direction for automatic generation of encoding_file
                    flag: --datain=%s
                    mutually_exclusive: encoding_file
                    requires: readout_times
encoding_file: (an existing file name)
                name of text file with PE directions/times
                flag: --datain=%s
                mutually_exclusive: encoding_direction
in_file: (an existing file name)
          name of 4D file with images
          flag: --imain=%s
readout_times: (a list of items which are a float)
                readout times (dwell times by # phase-encode steps minus 1)
                mutually_exclusive: encoding_file
                requires: encoding_direction

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
config: (a string, nipyype default value: b02b0.cnf)
        Name of config file specifying command line arguments
        flag: --config=%s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyype default value: {})
          Environment variables
estmov: (1 or 0)
        estimate movements if set
        flag: --estmov=%d
fwhm: (a float)
      FWHM (in mm) of gaussian smoothing kernel
      flag: --fwhm=%f
```

```
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interp: ('spline' or 'linear')
    Image interpolation model, linear or spline.
    flag: --interp=%s
max_iter: (an integer (int or long))
    max # of non-linear iterations
    flag: --miter=%d
minmet: (0 or 1)
    Minimisation method 0=Levenberg-Marquardt, 1=Scaled Conjugate
    Gradient
    flag: --minmet=%d
numprec: ('double' or 'float')
    Precision for representing Hessian, double or float.
    flag: --numprec=%s
out_base: (a file name)
    base-name of output files (spline coefficients (Hz) and movement
    parameters)
    flag: --out=%s
out_corrected: (a file name)
    name of 4D image file with unwarped images
    flag: --iout=%s
out_field: (a file name)
    name of image file with field (Hz)
    flag: --fout=%s
out_jac_prefix: (a unicode string, nipyte default value: jac)
    prefix for the warpfield images
    flag: --jacout=%s
out_logfile: (a file name)
    name of log-file
    flag: --logout=%s
out_warp_prefix: (a unicode string, nipyte default value: warpfield)
    prefix for the warpfield images (in mm)
    flag: --dfout=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
reg_lambda: (a float)
    lambda weighting value of the regularisation term
    flag: --miter=%0.f
regmod: ('bending_energy' or 'membrane_energy')
    Regularisation term implementation. Defaults to bending_energy. Note
    that the two functions have vastly different scales. The membrane
    energy is based on the first derivatives and the bending energy on
    the second derivatives. The second derivatives will typically be
    much smaller than the first derivatives, so input lambda will have
    to be larger for bending_energy to yield approximately the same
    level of regularisation.
    flag: --regmod=%s
regrid: (1 or 0)
    If set (=1), the calculations are done in a different grid
    flag: --regrid=%d
scale: (0 or 1)
    If set (=1), the images are individually scaled to a common mean
    flag: --scale=%d
splineorder: (an integer (int or long))
    order of spline, 2->Quadratic spline, 3->Cubic spline
```

```

    flag: --splineorder=%d
ssqlambda: (1 or 0)
    Weight lambda by the current value of the ssd. If used (=1), the
    effective weight of regularisation term becomes higher for the
    initial iterations, therefore initial steps are a little smoother
    than they would without weighting. This reduces the risk of finding
    a local minimum.
    flag: --ssqlambda=%d
subsamp: (an integer (int or long))
    sub-sampling scheme
    flag: --subsamp=%d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
warp_res: (a float)
    (approximate) resolution (in mm) of warp basis for the different
    sub-sampling levels.
    flag: --warpres=%f

```

Outputs:

```

out_corrected: (a file name)
    name of 4D image file with unwarped images
out_enc_file: (a file name)
    encoding directions file output for applytopup
out_field: (a file name)
    name of image file with field (Hz)
out_fieldcoef: (an existing file name)
    file containing the field coefficients
out_jacs: (a list of items which are an existing file name)
    Jacobian images
out_logfile: (a file name)
    name of log-file
out_movpar: (an existing file name)
    movpar.txt output file
out_warps: (a list of items which are an existing file name)
    warpfield images

```

References:: None

interfaces.fsl.fix

Classifier

[Link to code](#)

Wraps command **None**

Classify ICA components using a specific training dataset (<thresh> is in the range 0-100, typically 5-20).

Inputs:

```

[Mandatory]
thresh: (an integer (int or long))
    Threshold for cleanup.
    flag: %d, position: -1
trained_wts_file: (an existing file name)
    trained-weights file
    flag: %s, position: 2

```

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
artifacts_list_file: (a file name)
    Text file listing which ICs are artifacts; can be the output from
    classification or can be created manually
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mel_ica: (an existing directory name)
    Melodic output directory or directories
    flag: %s, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
artifacts_list_file: (a file name)
    Text file listing which ICs are artifacts; can be the output from
    classification or can be created manually
```

Cleaner

[Link to code](#)

Wraps command **None**

Extract features (for later training and/or classifying)

Inputs:

```
[Mandatory]
artifacts_list_file: (an existing file name)
    Text file listing which ICs are artifacts; can be the output from
    classification or can be created manually
    flag: %s, position: 1

[Optional]
aggressive: (a boolean)
    Apply aggressive (full variance) cleanup, instead of the default
    less-aggressive (unique variance) cleanup.
    flag: -A, position: 3
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cleanup_motion: (a boolean)
    cleanup motion confounds, looks for design.fsf for highpass filter
    cut-off
    flag: -m, position: 2
confound_file: (a file name)
    Include additional confound file.
    flag: -x %s, position: 4
confound_file_1: (a file name)
    Include additional confound file.
```



```

    flag: -x %s, position: 5
confound_file_2: (a file name)
    Include additional confound file.
    flag: -x %s, position: 6
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
highpass: (a float, nipyne default value: 100)
    cleanup motion confounds
    flag: -m -h %f, position: 2
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

cleaned_functional_file: (an existing file name)
    Cleaned session data

```

FeatureExtractor

[Link to code](#)**Wraps command None**

Extract features (for later training and/or classifying)

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mel_ica: (an existing directory name)
    Melodic output directory or directories
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

mel_ica: (an existing directory name)
    Melodic output directory or directories
    flag: %s, position: -1

```

Training

[Link to code](#)

Wraps command **None**

Train the classifier based on your own FEAT/MELODIC output directory.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
loo: (a boolean)
    full leave-one-out test with classifier training
    flag: -l, position: 2
mel_icas: (a list of items which are an existing directory name)
    Melodic output directories
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
trained_wts_filestem: (a unicode string)
    trained-weights filestem, used for trained_wts_file and output
    directories
    flag: %s, position: 1
```

Outputs:

```
trained_wts_file: (an existing file name)
    Trained-weights file
```

TrainingSetCreator

[Link to code](#)

Goes through set of provided melodic output directories, to find all the ones that have a hand_labels_noise.txt file in them.

This is outsourced as a separate class, so that the pipeline is rerun everytime a handlabeled file has been changed, or a new one created.

Inputs:

```
[Mandatory]

[Optional]
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mel_icas_in: (a list of items which are an existing directory name)
    Melodic output directories
    flag: %s, position: -1
```

Outputs:

```
mel_icas_out: (a list of items which are an existing directory name)
              Hand labels for noise vs signal
              flag: %s, position: -1
```

interfaces.fsl.maths**AR1Image**[Link to code](#)Wraps command **fslmaths**

Use fslmaths to generate an AR1 coefficient image across a given dimension. (Should use -odt float and probably demean first)

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
           dimension to find AR(1) coefficient across
           flag: -%sar1, position: 4
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
                   datatype to use for calculations (default is float)
                   flag: -dt %s, position: 1
nan2zeros: (a boolean)
            change NaNs to zeros before doing anything
            flag: -nan, position: 3
out_file: (a file name)
          image to write
          flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                  datatype to use for output (default uses input type)
                  flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          image written after calculations
```

References:: None

ApplyMask

[Link to code](#)

Wraps command **fslmaths**

Use fslmaths to apply a binary mask to another image.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         image to operate on
         flag: %s, position: 2
mask_file: (an existing file name)
           binary image defining mask space
           flag: -mas %s, position: 4

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipype default value: {})
         Environment variables
ignore_exception: (a boolean, nipype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
                   datatype to use for calculations (default is float)
                   flag: -dt %s, position: 1
nan2zeros: (a boolean)
            change NaNs to zeros before doing anything
            flag: -nan, position: 3
out_file: (a file name)
          image to write
          flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                 datatype to use for output (default uses input type)
                 flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          image written after calculations
```

References:: None

BinaryMaths

[Link to code](#)

Wraps command **fslmaths**

Use fslmaths to perform mathematical operations using a second image or a numeric value.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    image to operate on
    flag: %s, position: 2
operand_file: (an existing file name)
    second image to perform operation with
    flag: %s, position: 5
    mutually_exclusive: operand_value
operand_value: (a float)
    value to perform operation with
    flag: %.8f, position: 5
    mutually_exclusive: operand_file
operation: ('add' or 'sub' or 'mul' or 'div' or 'rem' or 'max' or
    'min')
    operation to perform
    flag: -%s, position: 4

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    flag: -dt %s, position: 1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    flag: -nan, position: 3
out_file: (a file name)
    image to write
    flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          image written after calculations
```

References:: None

ChangeData**Type**

[Link to code](#)

Wraps command **fslmaths**

Use **fslmaths** to change the datatype of an image.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         image to operate on
         flag: %s, position: 2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                 output data type
                 flag: -odt %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
                    datatype to use for calculations (default is float)
                    flag: -dt %s, position: 1
nan2zeros: (a boolean)
            change NaNs to zeros before doing anything
            flag: -nan, position: 3
out_file: (a file name)
           image to write
           flag: %s, position: -2
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          image written after calculations
```

References:: None

DilateImage

[Link to code](#)

Wraps command **fslmaths**

Use fslmaths to perform a spatial dilation of an image.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2
operation: ('mean' or 'modal' or 'max')
        filtering operation to perfoem in dilation
        flag: -dil%s, position: 6

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for calculations (default is float)
        flag: -dt %s, position: 1
kernel_file: (an existing file name)
        use external file for kernel
        flag: %s, position: 5
        mutually_exclusive: kernel_size
kernel_shape: ('3D' or '2D' or 'box' or 'boxv' or 'gauss' or 'sphere'
        or 'file')
        kernel shape to use
        flag: -kernel %s, position: 4
kernel_size: (a float)
        kernel size - voxels for box/boxv, mm for sphere, mm sigma for gauss
        flag: %.4f, position: 5
        mutually_exclusive: kernel_file
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        flag: -nan, position: 3
out_file: (a file name)
        image to write
        flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for output (default uses input type)
        flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_PAIR_GZ')
        FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```
out_file: (an existing file name)
          image written after calculations
```

References:: None**ErodeImage**[Link to code](#)**Wraps command `fslmaths`**Use `fslmaths` to perform a spatial erosion of an image.**Inputs:**

```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2

[Optional]
args: (a unicode string)
     Additional parameters to the command
     flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
                  datatype to use for calculations (default is float)
                  flag: -dt %s, position: 1
kernel_file: (an existing file name)
             use external file for kernel
             flag: %s, position: 5
             mutually_exclusive: kernel_size
kernel_shape: ('3D' or '2D' or 'box' or 'boxv' or 'gauss' or 'sphere'
              or 'file')
             kernel shape to use
             flag: -kernel %s, position: 4
kernel_size: (a float)
             kernel size - voxels for box/boxv, mm for sphere, mm sigma for gauss
             flag: %.4f, position: 5
             mutually_exclusive: kernel_file
minimum_filter: (a boolean, nipy default value: False)
               if true, minimum filter rather than erosion by zeroing-out
               flag: %s, position: 6
nan2zeros: (a boolean)
           change NaNs to zeros before doing anything
           flag: -nan, position: 3
out_file: (a file name)
         image to write
         flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                 datatype to use for output (default uses input type)
```



```

    flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:: None**IsotropicSmooth**[Link to code](#)**Wraps command `fslmaths`**Use `fslmaths` to spatially smooth an image with a gaussian kernel.**Inputs:**

```

[Mandatory]
fwhm: (a float)
    fwhm of smoothing kernel [mm]
    flag: -s %.5f, position: 4
    mutually_exclusive: sigma
in_file: (an existing file name)
    image to operate on
    flag: %s, position: 2
sigma: (a float)
    sigma of smoothing kernel [mm]
    flag: -s %.5f, position: 4
    mutually_exclusive: fwhm

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    flag: -dt %s, position: 1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    flag: -nan, position: 3
out_file: (a file name)
    image to write
    flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')

```

```

        datatype to use for output (default uses input type)
        flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
        FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:: None

MathsCommand[Link to code](#)Wraps command **fslmaths****Inputs:**

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
        datatype to use for calculations (default is float)
        flag: -dt %s, position: 1
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        flag: -nan, position: 3
out_file: (a file name)
        image to write
        flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                 or 'input')
        datatype to use for output (default uses input type)
        flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
        FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display

```

```
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          image written after calculations
```

References:: None

MaxImage[Link to code](#)Wraps command **fslmaths**

Use fslmaths to generate a max image across a given dimension.

Examples

```
>>> from nipy.interfaces.fsl.maths import MaxImage
>>> maxer = MaxImage()
>>> maxer.inputs.in_file = "functional.nii"
>>> maxer.dimension = "T"
>>> maxer.cmdline
'fslmaths functional.nii -Tmax functional_max.nii'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
          dimension to max across
          flag: -%smax, position: 4
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
                  datatype to use for calculations (default is float)
                  flag: -dt %s, position: 1
nan2zeros: (a boolean)
            change NaNs to zeros before doing anything
            flag: -nan, position: 3
out_file: (a file name)
          image to write
          flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                  datatype to use for output (default uses input type)
```

```

    flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
                        (default), `allatonce` - waits till command is finished to display
                        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
          image written after calculations

```

References:: None**MaxnImage**[Link to code](#)**Wraps command `fslmaths`**Use `fslmaths` to generate an image of index of max across a given dimension.**Inputs:**

```

[Mandatory]
in_file: (an existing file name)
         image to operate on
         flag: %s, position: 2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
           dimension to index max across
           flag: -%smaxn, position: 4
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
                   datatype to use for calculations (default is float)
                   flag: -dt %s, position: 1
nan2zeros: (a boolean)
            change NaNs to zeros before doing anything
            flag: -nan, position: 3
out_file: (a file name)
          image to write
          flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                  datatype to use for output (default uses input type)
                  flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type

```

```
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    image written after calculations
```

References:: None

MeanImage[Link to code](#)Wraps command **fslmaths**

Use fslmaths to generate a mean image across a given dimension.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    image to operate on
    flag: %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
    dimension to mean across
    flag: -%smean, position: 4
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    flag: -dt %s, position: 1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    flag: -nan, position: 3
out_file: (a file name)
    image to write
    flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          image written after calculations
```

References:: None

MedianImage[Link to code](#)Wraps command **fslmaths**

Use fslmaths to generate a median image across a given dimension.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
           dimension to median across
           flag: -%smedian, position: 4
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
                   datatype to use for calculations (default is float)
                   flag: -dt %s, position: 1
nan2zeros: (a boolean)
            change NaNs to zeros before doing anything
            flag: -nan, position: 3
out_file: (a file name)
          image to write
          flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                  datatype to use for output (default uses input type)
                  flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          image written after calculations
```

References:: None

MinImage

[Link to code](#)

Wraps command **fslmaths**

Use **fslmaths** to generate a minimum image across a given dimension.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
          image to operate on
          flag: %s, position: 2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
            dimension to min across
            flag: -%smin, position: 4
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
                    datatype to use for calculations (default is float)
                    flag: -dt %s, position: 1
nan2zeros: (a boolean)
            change NaNs to zeros before doing anything
            flag: -nan, position: 3
out_file: (a file name)
           image to write
           flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                  datatype to use for output (default uses input type)
                  flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          image written after calculations
```

References:: None

MultImageMaths

[Link to code](#)

Wraps command **fslmaths**

Use fslmaths to perform a sequence of mathematical operations.

Examples

```

>>> from nipy.interfaces.fsl import MultiImageMaths
>>> maths = MultiImageMaths()
>>> maths.inputs.in_file = "functional.nii"
>>> maths.inputs.op_string = "-add %s -mul -1 -div %s"
>>> maths.inputs.operand_files = ["functional2.nii", "functional3.nii"]
>>> maths.inputs.out_file = "functional4.nii"
>>> maths.cmdline
'fslmaths functional.nii -add functional2.nii -mul -1 -div functional3.nii_
↪functional4.nii'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2
op_string: (a string)
        python formatted string of operations to perform
        flag: %s, position: 4
operand_files: (a list of items which are an existing file name)
        list of file names to plug into op string

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for calculations (default is float)
        flag: -dt %s, position: 1
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        flag: -nan, position: 3
out_file: (a file name)
        image to write
        flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for output (default uses input type)
        flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_PAIR_GZ')
        FSL output type

```



```
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    image written after calculations
```

References:: None

PercentileImage

[Link to code](#)

Wraps command **fslmaths**

Use fslmaths to generate a percentile image across a given dimension.

Examples

```
>>> from nipy.interfaces.fsl.maths import MaxImage
>>> percer = PercentileImage()
>>> percer.inputs.in_file = "functional.nii"
>>> percer.dimension = "T"
>>> percer.perc = 90
>>> percer.cmdline
'fslmaths functional.nii -Tperc 90 functional_perc.nii'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    image to operate on
    flag: %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
    dimension to percentile across
    flag: -%sperc, position: 4
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    flag: -dt %s, position: 1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    flag: -nan, position: 3
out_file: (a file name)
    image to write
```

```

    flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
    datatype to use for output (default uses input type)
    flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
    FSL output type
perc: (0 <= a long integer <= 100)
    nth percentile (0-100) of FULL RANGE across dimension
    flag: %f, position: 5
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:: None

SpatialFilter[Link to code](#)Wraps command **fslmaths**

Use fslmaths to spatially filter an image.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    image to operate on
    flag: %s, position: 2
operation: ('mean' or 'median' or 'meanu')
    operation to filter with
    flag: -f%s, position: 6

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
    datatype to use for calculations (default is float)
    flag: -dt %s, position: 1
kernel_file: (an existing file name)
    use external file for kernel
    flag: %s, position: 5
    mutually_exclusive: kernel_size
kernel_shape: ('3D' or '2D' or 'box' or 'boxv' or 'gauss' or 'sphere'
               or 'file')

```

```

        kernel shape to use
        flag: -kernel %s, position: 4
kernel_size: (a float)
        kernel size - voxels for box/boxv, mm for sphere, mm sigma for gauss
        flag: %.4f, position: 5
        mutually_exclusive: kernel_file
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        flag: -nan, position: 3
out_file: (a file name)
        image to write
        flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
        datatype to use for output (default uses input type)
        flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
        FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:: None

StdImage[Link to code](#)Wraps command **fslmaths**Use **fslmaths** to generate a standard deviation in an image across a given dimension.**Inputs:**

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
        dimension to standard deviate across
        flag: -%sstd, position: 4
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')

```

```

        datatype to use for calculations (default is float)
        flag: -dt %s, position: 1
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        flag: -nan, position: 3
out_file: (a file name)
        image to write
        flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for output (default uses input type)
        flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_PAIR_GZ')
        FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:: None**TemporalFilter**[Link to code](#)**Wraps command `fslmaths`**Use `fslmaths` to apply a low, high, or bandpass temporal filter to a timeseries.**Inputs:**

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
highpass_sigma: (a float, nipy default value: -1)
        highpass filter sigma (in volumes)
        flag: -bptf %.6f, position: 4
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for calculations (default is float)
        flag: -dt %s, position: 1
lowpass_sigma: (a float, nipy default value: -1)
        lowpass filter sigma (in volumes)

```

```

        flag: %.6f, position: 5
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        flag: -nan, position: 3
out_file: (a file name)
        image to write
        flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for output (default uses input type)
        flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_PAIR_GZ')
        FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:: None

Threshold[Link to code](#)Wraps command **fslmaths**

Use fslmaths to apply a threshold to an image in a variety of ways.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2
thresh: (a float)
        threshold value
        flag: %s, position: 4

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
direction: ('below' or 'above', nipy default value: below)
        zero-out either below or above thresh value
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for calculations (default is float)
        flag: -dt %s, position: 1
nan2zeros: (a boolean)

```

```

        change NaNs to zeros before doing anything
        flag: -nan, position: 3
out_file: (a file name)
        image to write
        flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
        datatype to use for output (default uses input type)
        flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
        FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
use_nonzero_voxels: (a boolean)
        use nonzero voxels to calculate robust range
        requires: use_robust_range
use_robust_range: (a boolean)
        interpret thresh as percentage (0-100) of robust range

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:: None

UnaryMaths[Link to code](#)Wraps command **fslmaths**Use **fslmaths** to perform a variety of mathematical operations on an image.**Inputs:**

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2
operation: ('exp' or 'log' or 'sin' or 'cos' or 'tan' or 'asin' or
            'acos' or 'atan' or 'sqr' or 'sqrt' or 'recip' or 'abs' or 'bin' or
            'binv' or 'fillh' or 'fillh26' or 'index' or 'edge' or 'nan' or
            'nanm' or 'rand' or 'randn' or 'range')
        operation to perform
        flag: -%s, position: 4

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'

```

```

    or 'input')
    datatype to use for calculations (default is float)
    flag: -dt %s, position: 1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    flag: -nan, position: 3
out_file: (a file name)
    image to write
    flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    flag: -odt %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:: None

interfaces.fsl.model

Cluster

[Link to code](#)Wraps command **cluster**

Uses FSL cluster to perform clustering on statistical output

Examples

```

>>> cl = Cluster()
>>> cl.inputs.threshold = 2.3
>>> cl.inputs.in_file = 'zstat1.nii.gz'
>>> cl.inputs.out_localmax_txt_file = 'stats.txt'
>>> cl.inputs.use_mm = True
>>> cl.cmdline
'cluster --in=zstat1.nii.gz --olmax=stats.txt --thresh=2.3000000000 --mm'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input volume
    flag: --in=%s
threshold: (a float)
    threshold for input volume
    flag: --thresh=%.10f

[Optional]

```

```
args: (a unicode string)
    Additional parameters to the command
    flag: %s
connectivity: (an integer (int or long))
    the connectivity of voxels (default 26)
    flag: --connectivity=%d
cope_file: (a file name)
    cope volume
    flag: --cope=%s
dlh: (a float)
    smoothness estimate = sqrt(det(Lambda))
    flag: --dlh=%.10f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
find_min: (a boolean, nipy default value: False)
    find minima instead of maxima
    flag: --min
fractional: (a boolean, nipy default value: False)
    interprets the threshold as a fraction of the robust range
    flag: --fractional
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
minclustersize: (a boolean, nipy default value: False)
    prints out minimum significant cluster size
    flag: --minclustersize
no_table: (a boolean, nipy default value: False)
    suppresses printing of the table info
    flag: --no_table
num_maxima: (an integer (int or long))
    no of local maxima to report
    flag: --num=%d
out_index_file: (a boolean or a file name)
    output of cluster index (in size order)
    flag: --oindex=%s
out_localmax_txt_file: (a boolean or a file name)
    local maxima text file
    flag: --olmax=%s
out_localmax_vol_file: (a boolean or a file name)
    output of local maxima volume
    flag: --olmaxim=%s
out_max_file: (a boolean or a file name)
    filename for output of max image
    flag: --omax=%s
out_mean_file: (a boolean or a file name)
    filename for output of mean image
    flag: --omean=%s
out_pval_file: (a boolean or a file name)
    filename for image output of log pvals
    flag: --opvals=%s
out_size_file: (a boolean or a file name)
    filename for output of size image
    flag: --osize=%s
out_threshold_file: (a boolean or a file name)
    thresholded image
    flag: --othresh=%s
```



```

output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
peak_distance: (a float)
               minimum distance between local maxima/minima, in mm (default 0)
               flag: --peakdist=%.10f
pthreshold: (a float)
            p-threshold for clusters
            flag: --pthresh=%.10f
            requires: dlh, volume
std_space_file: (a file name)
                filename for standard-space volume
                flag: --stdvol=%s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
                 (default), `allatonce` - waits till command is finished to display
                 output, `file` - writes output to file, `none` - output is ignored
use_mm: (a boolean, nipy default value: False)
        use mm, not voxel, coordinates
        flag: --mm
volume: (an integer (int or long))
        number of voxels in the mask
        flag: --volume=%d
warpfield_file: (a file name)
                file containing warpfield
                flag: --warpvol=%s
xfm_file: (a file name)
           filename for Linear: input->standard-space transform. Non-linear:
           input->highres transform
           flag: --xfm=%s

```

Outputs:

```

index_file: (a file name)
            output of cluster index (in size order)
localmax_txt_file: (a file name)
                  local maxima text file
localmax_vol_file: (a file name)
                  output of local maxima volume
max_file: (a file name)
          filename for output of max image
mean_file: (a file name)
          filename for output of mean image
pval_file: (a file name)
          filename for image output of log pvals
size_file: (a file name)
          filename for output of size image
threshold_file: (a file name)
               thresholded image

```

References:: None

ContrastMgr[Link to code](#)Wraps command **contrast_mgr**Use FSL **contrast_mgr** command to evaluate contrasts

In interface mode this file assumes that all the required inputs are in the same location. This has deprecated for FSL versions 5.0.7+ as the necessary corrections file is no longer generated by FILMGLS.

Inputs:

```
[Mandatory]
corrections: (an existing file name)
    statistical corrections used within FILM modelling
dof_file: (an existing file name)
    degrees of freedom
param_estimates: (a list of items which are an existing file name)
    Parameter estimates for each column of the design matrix
sigmasquareds: (an existing file name)
    summary of residuals, See Woolrich, et. al., 2001
tcon_file: (an existing file name)
    contrast file containing T-contrasts
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
contrast_num: (a long integer >= 1)
    contrast number to start labeling copes from
    flag: -cope
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fcon_file: (an existing file name)
    contrast file containing F-contrasts
    flag: -f %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
suffix: (a unicode string)
    suffix to put on the end of the cope filename before the contrast
    number, default is nothing
    flag: -suffix %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
copes: (a list of items which are an existing file name)
    Contrast estimates for each contrast
fstats: (a list of items which are an existing file name)
    f-stat file for each contrast
neffs: (a list of items which are an existing file name)
    neff file ?? for each contrast
tstats: (a list of items which are an existing file name)
    t-stat file for each contrast
varcopes: (a list of items which are an existing file name)
    Variance estimates for each contrast
zfstats: (a list of items which are an existing file name)
    z-stat file for each F contrast
zstats: (a list of items which are an existing file name)
```

```
z-stat file for each contrast
```

References:: None

FEAT

[Link to code](#)

Wraps command **feat**

Uses FSL feat to calculate first level stats

Inputs:

```
[Mandatory]
fsf_file: (an existing file name)
    File specifying the feat design spec file
    flag: %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
feat_dir: (an existing directory name)
```

References:: None

FEATModel

[Link to code](#)

Wraps command **feat_model**

Uses FSL feat_model to generate design.mat files

Inputs:

```
[Mandatory]
ev_files: (a list of items which are an existing file name)
    Event spec files generated by level1design
    flag: %s, position: 1
fsf_file: (an existing file name)
    File specifying the feat design spec file
    flag: %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
```

```
flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipyne default value: {})
         Environment variables
ignore_exception: (a boolean, nipyne default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
con_file: (an existing file name)
           Contrast file containing contrast vectors
design_cov: (an existing file name)
            Graphical representation of design covariance
design_file: (an existing file name)
            Mat file containing ascii matrix for design
design_image: (an existing file name)
             Graphical representation of design matrix
fcon_file: (a file name)
           Contrast file containing contrast vectors
```

References:: None

FEATRegister

[Link to code](#)

Register feat directories to a specific standard

Inputs:

```
[Mandatory]
feat_dirs: (a list of items which are an existing directory name)
           Lower level feat dirs
reg_image: (an existing file name)
           image to register to (will be treated as standard)

[Optional]
ignore_exception: (a boolean, nipyne default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
reg_dof: (an integer (int or long), nipyne default value: 12)
          registration degrees of freedom
```

Outputs:

```
fsf_file: (an existing file name)
          FSL feat specification file
```

FILMGLS

[Link to code](#)Wraps command **film_gls**

Use FSL `film_gls` command to fit a design matrix to voxel timeseries

Examples

Initialize with no options, assigning them when calling run:

```
>>> from nipy.interfaces import fsl
>>> fgls = fsl.FILMGLS()
>>> res = fgls.run('in_file', 'design_file', 'thresh', rn='stats')
```

Assign options through the inputs attribute:

```
>>> fgls = fsl.FILMGLS()
>>> fgls.inputs.in_file = 'functional.nii'
>>> fgls.inputs.design_file = 'design.mat'
>>> fgls.inputs.threshold = 10
>>> fgls.inputs.results_dir = 'stats'
>>> res = fgls.run()
```

Specify options when creating an instance:

```
>>> fgls = fsl.FILMGLS(in_file='functional.nii', design_file='design.mat',
↳threshold=10, results_dir='stats')
>>> res = fgls.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input data file
        flag: %s, position: -3

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
autocorr_estimate_only: (a boolean)
                        perform autocorrelation estimation only
                        flag: -ac
                        mutually_exclusive: autocorr_estimate_only, fit_armodel,
                        tukey_window, multitaper_product, use_pava, autocorr_noestimate
autocorr_noestimate: (a boolean)
                     do not estimate autocorrs
                     flag: -noest
                     mutually_exclusive: autocorr_estimate_only, fit_armodel,
                     tukey_window, multitaper_product, use_pava, autocorr_noestimate
brightness_threshold: (a long integer >= 0)
                      susan brightness threshold, otherwise it is estimated
                      flag: -epith %d
design_file: (an existing file name)
             design matrix file
             flag: %s, position: -2
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
fit_armodel: (a boolean)
             fits autoregressive model - default is to use tukey with
             M=sqrt(numvols)
             flag: -ar
```

```

    mutually_exclusive: autocorr_estimate_only, fit_armodel,
        tukey_window, multitaper_product, use_pava, autocorr_noestimate
full_data: (a boolean)
    output full data
    flag: -v
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_size: (an integer (int or long))
    susan mask size
    flag: -ms %d
multitaper_product: (an integer (int or long))
    multitapering with slepian tapers and num is the time-bandwidth
    product
    flag: -mt %d
    mutually_exclusive: autocorr_estimate_only, fit_armodel,
        tukey_window, multitaper_product, use_pava, autocorr_noestimate
output_pwdata: (a boolean)
    output prewhitened data and average design matrix
    flag: -output_pwdata
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
results_dir: (a directory name, nipy default value: results)
    directory to store results in
    flag: -rn %s
smooth_autocorr: (a boolean)
    Smooth auto corr estimates
    flag: -sa
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (a floating point number >= 0.0, nipy default value:
    0.0)
    threshold
    flag: %f, position: -1
tukey_window: (an integer (int or long))
    tukey window size to estimate autocorr
    flag: -tukey %d
    mutually_exclusive: autocorr_estimate_only, fit_armodel,
        tukey_window, multitaper_product, use_pava, autocorr_noestimate
use_pava: (a boolean)
    estimates autocorr using PAVA
    flag: -pava

```

Outputs:

```

corrections: (an existing file name)
    statistical corrections used within FILM modeling
dof_file: (an existing file name)
    degrees of freedom
logfile: (an existing file name)
    FILM run logfile
param_estimates: (a list of items which are an existing file name)
    Parameter estimates for each column of the design matrix
residual4d: (an existing file name)
    Model fit residual mean-squared error for each time point

```

```

results_dir: (an existing directory name)
             directory storing model estimation output
sigmasquareds: (an existing file name)
               summary of residuals, See Woolrich, et. al., 2001
thresholdac: (an existing file name)
             The FILM autocorrelation parameters

```

References:: None

FLAMEO

[Link to code](#)

Wraps command **flameo**

Use FSL flameo command to perform higher level model fits

Examples

Initialize FLAMEO with no options, assigning them when calling run:

```

>>> from nipy.interfaces import fsl
>>> flameo = fsl.FLAMEO()
>>> flameo.inputs.cope_file = 'cope.nii.gz'
>>> flameo.inputs.var_cope_file = 'varcope.nii.gz'
>>> flameo.inputs.cov_split_file = 'cov_split.mat'
>>> flameo.inputs.design_file = 'design.mat'
>>> flameo.inputs.t_con_file = 'design.con'
>>> flameo.inputs.mask_file = 'mask.nii'
>>> flameo.inputs.run_mode = 'fe'
>>> flameo.cmdline
'flameo --copefile=cope.nii.gz --covsplitfile=cov_split.mat --designfile=design.
↪mat --ld=stats --maskfile=mask.nii --runmode=fe --tcontrastsfile=design.con --
↪varcopefile=varcope.nii.gz'

```

Inputs:

```

[Mandatory]
cope_file: (an existing file name)
           cope regressor data file
           flag: --copefile=%s
cov_split_file: (an existing file name)
               ascii matrix specifying the groups the covariance is split into
               flag: --covsplitfile=%s
design_file: (an existing file name)
            design matrix file
            flag: --designfile=%s
mask_file: (an existing file name)
           mask file
           flag: --maskfile=%s
run_mode: ('fe' or 'ols' or 'flame1' or 'flame12')
           inference to perform
           flag: --runmode=%s
t_con_file: (an existing file name)
            ascii matrix specifying t-contrasts
            flag: --tcontrastsfile=%s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s

```

```
burnin: (an integer (int or long))
    number of jumps at start of mcmc to be discarded
    flag: --burnin=%d
dof_var_cope_file: (an existing file name)
    dof data file for varcope data
    flag: --dofvarcopefile=%s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
f_con_file: (an existing file name)
    ascii matrix specifying f-contrasts
    flag: --fcontrastsfile=%s
fix_mean: (a boolean)
    fix mean for tfit
    flag: --fixmean
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
infer_outliers: (a boolean)
    infer outliers - not for fe
    flag: --inferoutliers
log_dir: (a directory name, nipyre default value: stats)
    flag: --ld=%s
n_jumps: (an integer (int or long))
    number of jumps made by mcmc
    flag: --njumps=%d
no_pe_outputs: (a boolean)
    do not output pe files
    flag: --nopeoutput
outlier_iter: (an integer (int or long))
    Number of max iterations to use when inferring outliers. Default is
    12.
    flag: --ioni=%d
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
sample_every: (an integer (int or long))
    number of jumps for each sample
    flag: --sampleevery=%d
sigma_dofs: (an integer (int or long))
    sigma (in mm) to use for Gaussian smoothing the DOFs in FLAME 2.
    Default is 1mm, -1 indicates no smoothing
    flag: --sigma_dofs=%d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
var_cope_file: (an existing file name)
    varcope weightings data file
    flag: --varcopefile=%s
```

Outputs:

```
copes: (a list of items which are an existing file name)
    Contrast estimates for each contrast
fstats: (a list of items which are an existing file name)
    f-stat file for each contrast
```



```

mrefvars: (a list of items which are an existing file name)
           mean random effect variances for each contrast
pes: (a list of items which are an existing file name)
      Parameter estimates for each column of the design matrix for each
      voxel
res4d: (a list of items which are an existing file name)
        Model fit residual mean-squared error for each time point
stats_dir: (a directory name)
            directory storing model estimation output
tdof: (a list of items which are an existing file name)
        temporal dof file for each contrast
tstats: (a list of items which are an existing file name)
          t-stat file for each contrast
var_copes: (a list of items which are an existing file name)
            Variance estimates for each contrast
weights: (a list of items which are an existing file name)
           weights file for each contrast
zfstats: (a list of items which are an existing file name)
           z stat file for each f contrast
zstats: (a list of items which are an existing file name)
          z-stat file for each contrast

```

References:: None None

GLM

[Link to code](#)

Wraps command `fsl_glm`

FSL GLM:

Example

```

>>> import nipy.interfaces.fsl as fsl
>>> glm = fsl.GLM(in_file='functional.nii', design='maps.nii', output_type='NIFTI
↪')
>>> glm.cmdline
'fsl_glm -i functional.nii -d maps.nii -o functional_glm.nii'

```

Inputs:

```

[Mandatory]
design: (an existing file name)
        file name of the GLM design matrix (text time courses for temporal
        regression or an image file for spatial regression)
        flag: -d %s, position: 2
in_file: (an existing file name)
          input file name (text matrix or 3D/4D image file)
          flag: -i %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
contrasts: (an existing file name)
            matrix of t-statics contrasts
            flag: -c %s
dat_norm: (a boolean)
            switch on normalization of the data time series to unit std

```

```
    deviation
    flag: --dat_norm
demean: (a boolean)
    switch on demeaning of design and data
    flag: --demean
des_norm: (a boolean)
    switch on normalization of the design matrix columns to unit std
    deviation
    flag: --des_norm
dof: (an integer (int or long))
    set degrees of freedom explicitly
    flag: --dof=%d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    mask image file name if input is image
    flag: -m %s
out_cope: (a file name)
    output file name for COPE (either as txt or image
    flag: --out_cope=%s
out_data_name: (a file name)
    output file name for pre-processed data
    flag: --out_data=%s
out_f_name: (a file name)
    output file name for F-value of full model fit
    flag: --out_f=%s
out_file: (a file name)
    filename for GLM parameter estimates (GLM betas)
    flag: -o %s, position: 3
out_p_name: (a file name)
    output file name for p-values of Z-stats (either as text file or
    image)
    flag: --out_p=%s
out_pf_name: (a file name)
    output file name for p-value for full model fit
    flag: --out_pf=%s
out_res_name: (a file name)
    output file name for residuals
    flag: --out_res=%s
out_sigsq_name: (a file name)
    output file name for residual noise variance sigma-square
    flag: --out_sigsq=%s
out_t_name: (a file name)
    output file name for t-stats (either as txt or image
    flag: --out_t=%s
out_varcb_name: (a file name)
    output file name for variance of COPEs
    flag: --out_varcb=%s
out_vnscales_name: (a file name)
    output file name for scaling factors for variance normalisation
    flag: --out_vnscales=%s
out_z_name: (a file name)
    output file name for Z-stats (either as txt or image
```

```

    flag: --out_z=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
var_norm: (a boolean)
    perform MELODIC variance-normalisation on data
flag: --vn

```

Outputs:

```

out_cope: (a list of items which are an existing file name)
    output file name for COPEs (either as text file or image)
out_data: (a list of items which are an existing file name)
    output file for preprocessed data
out_f: (a list of items which are an existing file name)
    output file name for F-value of full model fit
out_file: (an existing file name)
    file name of GLM parameters (if generated)
out_p: (a list of items which are an existing file name)
    output file name for p-values of Z-stats (either as text file or
    image)
out_pf: (a list of items which are an existing file name)
    output file name for p-value for full model fit
out_res: (a list of items which are an existing file name)
    output file name for residuals
out_sigsq: (a list of items which are an existing file name)
    output file name for residual noise variance sigma-square
out_t: (a list of items which are an existing file name)
    output file name for t-stats (either as text file or image)
out_varcb: (a list of items which are an existing file name)
    output file name for variance of COPEs
out_vnscales: (a list of items which are an existing file name)
    output file name for scaling factors for variance normalisation
out_z: (a list of items which are an existing file name)
    output file name for COPEs (either as text file or image)

```

References:: None

L2Model[Link to code](#)

Generate subject specific second level model

Examples

```

>>> from nipy.interfaces.fsl import L2Model
>>> model = L2Model(num_copes=3) # 3 sessions

```

Inputs:

```

[Mandatory]
num_copes: (a long integer >= 1)
    number of copes to be combined

```

```
[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

Outputs:

```
design_con: (an existing file name)
    design contrast file
design_grp: (an existing file name)
    design group file
design_mat: (an existing file name)
    design matrix file
```

Level1Design[Link to code](#)

Generate FEAT specific files

Examples

```
>>> level1design = Level1Design()
>>> level1design.inputs.interscan_interval = 2.5
>>> level1design.inputs.bases = {'dgamma':{'derivs': False}}
>>> level1design.inputs.session_info = 'session_info.npz'
>>> level1design.run()
```

Inputs:

```
[Mandatory]
bases: (a dictionary with keys which are 'dgamma' and with values
    which are a dictionary with keys which are 'derivs' and with values
    which are a boolean or a dictionary with keys which are 'gamma' and
    with values which are a dictionary with keys which are 'derivs' or
    'gamma_sigma' or 'gamma_delay' and with values which are any value or
    a dictionary with keys which are 'custom' and with values which are
    a dictionary with keys which are 'bfcustompath' and with values
    which are a unicode string or a dictionary with keys which are
    'none' and with values which are a dictionary with keys which are
    any value and with values which are any value or a dictionary with
    keys which are 'none' and with values which are None)
    name of basis function and options e.g., {'dgamma': {'derivs':
    True}}
interscan_interval: (a float)
    Interscan interval (in secs)
model_serial_correlations: (a boolean)
    Option to model serial correlations using an autoregressive
    estimator (order 1). Setting this option is only useful in the
    context of the fsf file. If you set this to False, you need to
    repeat this option for FILMGLS by setting autocorr_noestimate to
    True
session_info: (any value)
    Session specific information generated by ``modelgen.SpecifyModel``

[Optional]
contrasts: (a list of items which are a tuple of the form: (a unicode
    string, 'T', a list of items which are a unicode string, a list of
```

```

    items which are a float) or a tuple of the form: (a unicode string,
    'T', a list of items which are a unicode string, a list of items
    which are a float, a list of items which are a float) or a tuple of
    the form: (a unicode string, 'F', a list of items which are a tuple
    of the form: (a unicode string, 'T', a list of items which are a
    unicode string, a list of items which are a float) or a tuple of
    the form: (a unicode string, 'T', a list of items which are a
    unicode string, a list of items which are a float, a list of items
    which are a float)))
    List of contrasts with each contrast being a list of the form -
    [('name', 'stat', [condition list], [weight list], [session list])].
    if session list is None or not provided, all sessions are used. For
    F contrasts, the condition list should contain previously defined
    T-contrasts.
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
orthogonalization: (a dictionary with keys which are an integer (int
    or long) and with values which are a dictionary with keys which are
    an integer (int or long) and with values which are a boolean or an
    integer (int or long))
    which regressors to make orthogonal e.g., {1: {0:0,1:0,2:0}, 2:
    {0:1,1:1,2:0}} to make the second regressor in a 2-regressor model
    orthogonal to the first.

```

Outputs:

```

ev_files: (a list of items which are a list of items which are an
    existing file name)
    condition information files
fsf_files: (a list of items which are an existing file name)
    FSL feat specification files

```

MELODIC

[Link to code](#)

Wraps command **melodic**

Multivariate Exploratory Linear Optimised Decomposition into Independent Components

Examples

```

>>> melodic_setup = MELODIC()
>>> melodic_setup.inputs.approach = 'tica'
>>> melodic_setup.inputs.in_files = ['functional.nii', 'functional2.nii',
↪ 'functional3.nii']
>>> melodic_setup.inputs.no_bet = True
>>> melodic_setup.inputs.bg_threshold = 10
>>> melodic_setup.inputs.tr_sec = 1.5
>>> melodic_setup.inputs.mm_thresh = 0.5
>>> melodic_setup.inputs.out_stats = True
>>> melodic_setup.inputs.t_des = 'timeDesign.mat'
>>> melodic_setup.inputs.t_con = 'timeDesign.con'
>>> melodic_setup.inputs.s_des = 'subjectDesign.mat'
>>> melodic_setup.inputs.s_con = 'subjectDesign.con'
>>> melodic_setup.inputs.out_dir = 'groupICA.out'
>>> melodic_setup.cmdline

```

```
'melodic -i functional.nii,functional2.nii,functional3.nii -a tica --
↳bgthreshold=10.000000 --mmthresh=0.500000 --nobet -o groupICA.out --Ostats --
↳Scon=subjectDesign.con --Sdes=subjectDesign.mat --Tcon=timeDesign.con --
↳Tdes=timeDesign.mat --tr=1.500000'
>>> melodic_setup.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
          input file names (either single file name or a list)
          flag: -i %s, position: 0

[Optional]
ICs: (an existing file name)
     filename of the IC components file for mixture modelling
     flag: --ICs=%s
approach: (a unicode string)
          approach for decomposition, 2D: defl, symm (default), 3D: tica
          (default), concat
          flag: -a %s
args: (a unicode string)
      Additional parameters to the command
      flag: %s
bg_image: (an existing file name)
          specify background image for report (default: mean image)
          flag: --bgimage=%s
bg_threshold: (a float)
              brain/non-brain threshold used to mask non-brain voxels, as a
              percentage (only if --nobet selected)
              flag: --bgthreshold=%f
cov_weight: (a float)
            voxel-wise weights for the covariance matrix (e.g. segmentation
            information)
            flag: --covarweight=%f
dim: (an integer (int or long))
     dimensionality reduction into #num dimensions (default: automatic
     estimation)
     flag: -d %d
dim_est: (a unicode string)
         use specific dim. estimation technique: lap, bic, mdl, aic, mean
         (default: lap)
         flag: --dimest=%s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
epsilon: (a float)
         minimum error change
         flag: --eps=%f
epsilonS: (a float)
          minimum error change for rank-1 approximation in TICA
          flag: --epsS=%f
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
log_power: (a boolean)
            calculate log of power for frequency spectrum
```

```

    flag: --logPower
mask: (an existing file name)
    file name of mask for thresholding
    flag: -m %s
max_restart: (an integer (int or long))
    maximum number of restarts
    flag: --maxrestart=%d
maxit: (an integer (int or long))
    maximum number of iterations before restart
    flag: --maxit=%d
migp: (a boolean)
    switch on MIGP data reduction
    flag: --migp
migpN: (an integer (int or long))
    number of internal Eigenmaps
    flag: --migpN %d
migp_factor: (an integer (int or long))
    Internal Factor of mem-threshold relative to number of Eigenmaps
    (default: 2)
    flag: --migp_factor %d
migp_shuffle: (a boolean)
    randomise MIGP file order (default: TRUE)
    flag: --migp_shuffle
mix: (an existing file name)
    mixing matrix for mixture modelling / filtering
    flag: --mix=%s
mm_thresh: (a float)
    threshold for Mixture Model based inference
    flag: --mmthresh=%f
no_bet: (a boolean)
    switch off BET
    flag: --nobet
no_mask: (a boolean)
    switch off masking
    flag: --nomask
no_mm: (a boolean)
    switch off mixture modelling on IC maps
    flag: --no_mm
non_linearity: (a unicode string)
    nonlinearity: gauss, tanh, pow3, pow4
    flag: --nl=%s
num_ICs: (an integer (int or long))
    number of IC's to extract (for deflation approach)
    flag: -n %d
out_all: (a boolean)
    output everything
    flag: --Oall
out_dir: (a directory name)
    output directory name
    flag: -o %s
out_mean: (a boolean)
    output mean volume
    flag: --Omean
out_orig: (a boolean)
    output the original ICs
    flag: --Oorig
out_pca: (a boolean)
    output PCA results

```

```

    flag: --Opc
out_stats: (a boolean)
    output thresholded maps and probability maps
    flag: --Ostats
out_unmix: (a boolean)
    output unmixing matrix
    flag: --Ounmix
out_white: (a boolean)
    output whitening/dewhitening matrices
    flag: --Owhite
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
pbsc: (a boolean)
    switch off conversion to percent BOLD signal change
    flag: --pbsc
rem_cmp: (a list of items which are an integer (int or long))
    component numbers to remove
    flag: -f %d
remove_deriv: (a boolean)
    removes every second entry in paradigm file (EV derivatives)
    flag: --remove_deriv
report: (a boolean)
    generate Melodic web report
    flag: --report
report_maps: (a unicode string)
    control string for spatial map images (see slicer)
    flag: --report_maps=%s
s_con: (an existing file name)
    t-contrast matrix across subject-domain
    flag: --Scon=%s
s_des: (an existing file name)
    design matrix across subject-domain
    flag: --Sdes=%s
sep_vn: (a boolean)
    switch off joined variance normalization
    flag: --sep_vn
sep_whiten: (a boolean)
    switch on separate whitening
    flag: --sep_whiten
smode: (an existing file name)
    matrix of session modes for report generation
    flag: --smode=%s
t_con: (an existing file name)
    t-contrast matrix across time-domain
    flag: --Tcon=%s
t_des: (an existing file name)
    design matrix across time-domain
    flag: --Tdes=%s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tr_sec: (a float)
    TR in seconds
    flag: --tr=%f
update_mask: (a boolean)
    switch off mask updating

```



```

    flag: --update_mask
var_norm: (a boolean)
    switch off variance normalization
    flag: --vn

```

Outputs:

```

out_dir: (an existing directory name)
report_dir: (an existing directory name)

```

References:: None

MultipleRegressDesign

[Link to code](#)

Generate multiple regression design

Note: FSL does not demean columns for higher level analysis.

Please see [FSL documentation](#) for more details on model specification for higher level analysis.

Examples

```

>>> from nipyre.interfaces.fsl import MultipleRegressDesign
>>> model = MultipleRegressDesign()
>>> model.inputs.contrasts = [['group mean', 'T', ['reg1'], [1]]]
>>> model.inputs.regressors = dict(reg1=[1, 1, 1], reg2=[2., -4, 3])
>>> model.run()

```

Inputs:

```

[Mandatory]
contrasts: (a list of items which are a tuple of the form: (a unicode
    string, 'T', a list of items which are a unicode string, a list of
    items which are a float) or a tuple of the form: (a unicode string,
    'F', a list of items which are a tuple of the form: (a unicode
    string, 'T', a list of items which are a unicode string, a list of
    items which are a float)))
    List of contrasts with each contrast being a list of the form -
    [['name', 'stat', [condition list], [weight list]]]. if session list
    is None or not provided, all sessions are used. For F contrasts, the
    condition list should contain previously defined T-contrasts without
    any weight list.
regressors: (a dictionary with keys which are a unicode string and
    with values which are a list of items which are a float)
    dictionary containing named lists of regressors

[Optional]
groups: (a list of items which are an integer (int or long))
    list of group identifiers (defaults to single group)
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run

```

Outputs:

```

design_con: (an existing file name)
    design t-contrast file

```

```

design_fts: (an existing file name)
            design f-contrast file
design_grp: (an existing file name)
            design group file
design_mat: (an existing file name)
            design matrix file

```

Randomise

[Link to code](#)

Wraps command **randomise**

FSL Randomise: feeds the 4D projected FA data into GLM modelling and thresholding in order to find voxels which correlate with your model

Example

```

>>> import nipy.interfaces.fsl as fsl
>>> rand = fsl.Randomise(in_file='allFA.nii', mask = 'mask.nii', tcon='design.con
↳', design_mat='design.mat')
>>> rand.cmdline
'randomise -i allFA.nii -o "tbss_" -d design.mat -t design.con -m mask.nii'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         4D input file
         flag: -i %s, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
base_name: (a unicode string, nipy default value: tbss_)
           the rootname that all generated files will have
           flag: -o "%s", position: 1
c_thresh: (a float)
          carry out cluster-based thresholding
          flag: -c %.2f
cm_thresh: (a float)
           carry out cluster-mass-based thresholding
           flag: -C %.2f
demean: (a boolean)
        demean data temporally before model fitting
        flag: -D
design_mat: (an existing file name)
           design matrix file
           flag: -d %s, position: 2
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
f_c_thresh: (a float)
           carry out f cluster thresholding
           flag: -F %.2f
f_cm_thresh: (a float)
            carry out f cluster-mass thresholding

```

```

    flag: -S %.2f
f_only: (a boolean)
    calculate f-statistics only
    flag: --f_only
fcon: (an existing file name)
    f contrasts file
    flag: -f %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    mask image
    flag: -m %s
num_perm: (an integer (int or long))
    number of permutations (default 5000, set to 0 for exhaustive)
    flag: -n %d
one_sample_group_mean: (a boolean)
    perform 1-sample group-mean test instead of generic permutation test
    flag: -l
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
p_vec_n_dist_files: (a boolean)
    output permutation vector and null distribution text files
    flag: -P
raw_stats_imgs: (a boolean)
    output raw ( unpermuted ) statistic images
    flag: -R
seed: (an integer (int or long))
    specific integer seed for random number generator
    flag: --seed=%d
show_info_parallel_mode: (a boolean)
    print out information required for parallel mode and exit
    flag: -Q
show_total_perms: (a boolean)
    print out how many unique permutations would be generated and exit
    flag: -q
tcon: (an existing file name)
    t contrasts file
    flag: -t %s, position: 3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tfce: (a boolean)
    carry out Threshold-Free Cluster Enhancement
    flag: -T
tfce2D: (a boolean)
    carry out Threshold-Free Cluster Enhancement with 2D optimisation
    flag: --T2
tfce_C: (a float)
    TFCE connectivity (6 or 26; default=6)
    flag: --tfce_C=%.2f
tfce_E: (a float)
    TFCE extent parameter (default=0.5)
    flag: --tfce_E=%.2f
tfce_H: (a float)
    TFCE height parameter (default=2)

```

```
    flag: --tfce_H=%.2f
var_smooth: (an integer (int or long))
    use variance smoothing (std is in mm)
    flag: -v %d
vox_p_values: (a boolean)
    output voxelwise (corrected and uncorrected) p-value images
    flag: -x
x_block_labels: (an existing file name)
    exchangeability block labels file
    flag: -e %s
```

Outputs:

```
f_corrected_p_files: (a list of items which are an existing file
    name)
    f contrast FWE (Family-wise error) corrected p values files
f_p_files: (a list of items which are an existing file name)
    f contrast uncorrected p values files
fstat_files: (a list of items which are an existing file name)
    f contrast raw statistic
t_corrected_p_files: (a list of items which are an existing file
    name)
    t contrast FWE (Family-wise error) corrected p values files
t_p_files: (a list of items which are an existing file name)
    f contrast uncorrected p values files
tstat_files: (a list of items which are an existing file name)
    t contrast raw statistic
```

References:: None

SMM

[Link to code](#)

Wraps command **mm -ld=logdir**

Spatial Mixture Modelling. For more detail on the spatial mixture modelling see Mixture Models with Adaptive Spatial Regularisation for Segmentation with an Application to FMRI Data; Woolrich, M., Behrens, T., Beckmann, C., and Smith, S.; IEEE Trans. Medical Imaging, 24(1):1-11, 2005.

Inputs:

```
[Mandatory]
mask: (an existing file name)
    mask file
    flag: --mask="%s", position: 1
spatial_data_file: (an existing file name)
    statistics spatial map
    flag: --sdf="%s", position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

```

no_deactivation_class: (a boolean)
    enforces no deactivation class
    flag: --zfstatmode, position: 2
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

activation_p_map: (an existing file name)
deactivation_p_map: (an existing file name)
null_p_map: (an existing file name)

```

References:: None

SmoothEstimate[Link to code](#)Wraps command **smoothest**

Estimates the smoothness of an image

Examples

```

>>> est = SmoothEstimate()
>>> est.inputs.zstat_file = 'zstat1.nii.gz'
>>> est.inputs.mask_file = 'mask.nii'
>>> est.cmdline
'smoosthest --mask=mask.nii --zstat=zstat1.nii.gz'

```

Inputs:

```

[Mandatory]
dof: (an integer (int or long))
    number of degrees of freedom
    flag: --dof=%d
    mutually_exclusive: zstat_file
mask_file: (an existing file name)
    brain mask volume
    flag: --mask=%s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type

```

```

residual_fit_file: (an existing file name)
    residual-fit image file
    flag: --res=%s
    requires: dof
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
zstat_file: (an existing file name)
    zstat image file
    flag: --zstat=%s
    mutually_exclusive: dof

```

Outputs:

```

dlh: (a float)
    smoothness estimate sqrt(det(Lambda))
resels: (a float)
    number of resels
volume: (an integer (int or long))
    number of voxels in mask

```

References:: None

interfaces.fsl.possum

B0Calc

[Link to code](#)Wraps command **b0calc**

B0 inhomogeneities occur at interfaces of materials with different magnetic susceptibilities, such as tissue-air interfaces. These differences lead to distortion in the local magnetic field, as Maxwell's equations need to be satisfied. An example of B0 inhomogeneity is the first volume of the 4D volume ``$FSLDIR/data/possum/b0_ppm.nii.gz``.

Examples

```

>>> from nipy.interfaces.fsl import B0Calc
>>> b0calc = B0Calc()
>>> b0calc.inputs.in_file = 'tissue+air_map.nii'
>>> b0calc.inputs.z_b0 = 3.0
>>> b0calc.inputs.output_type = "NIFTI_GZ"
>>> b0calc.cmdline
'b0calc -i tissue+air_map.nii -o tissue+air_map_b0field.nii.gz --b0=3.00'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    filename of input image (usually a tissue/air segmentation)
    flag: -i %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
chi_air: (a float)

```

```

    susceptibility of air
    flag: --chi0=%e
compute_xyz: (a boolean)
    calculate and save all 3 field components (i.e. x,y,z)
    flag: --xyz
delta: (a float)
    Delta value (chi_tissue - chi_air)
    flag: -d %e
directconv: (a boolean)
    use direct (image space) convolution, not FFT
    flag: --directconv
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
extendboundary: (a float)
    Relative proportion to extend voxels at boundary
    flag: --extendboundary=%0.2f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    filename of B0 output volume
    flag: -o %s, position: 1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
x_b0: (a float)
    Value for zeroth-order b0 field (x-component), in Tesla
    flag: --b0x=%0.2f
    mutually_exclusive: xyz_b0
x_grad: (a float)
    Value for zeroth-order x-gradient field (per mm)
    flag: --gx=%0.4f
xyz_b0: (a tuple of the form: (a float, a float, a float))
    Zeroth-order B0 field in Tesla
    flag: --b0x=%0.2f --b0y=%0.2f --b0z=%0.2f
    mutually_exclusive: x_b0, y_b0, z_b0
y_b0: (a float)
    Value for zeroth-order b0 field (y-component), in Tesla
    flag: --b0y=%0.2f
    mutually_exclusive: xyz_b0
y_grad: (a float)
    Value for zeroth-order y-gradient field (per mm)
    flag: --gy=%0.4f
z_b0: (a float)
    Value for zeroth-order b0 field (z-component), in Tesla
    flag: --b0z=%0.2f
    mutually_exclusive: xyz_b0
z_grad: (a float)
    Value for zeroth-order z-gradient field (per mm)
    flag: --gz=%0.4f

```

Outputs:

```
out_file: (an existing file name)
          filename of B0 output volume
```

References:: None

interfaces.fsl.preprocess

ApplyWarp

[Link to code](#)

Wraps command **applywarp**

Use FSL's applywarp to apply the results of a FNIRT registration

Examples

```
>>> from nipyype.interfaces import fsl
>>> from nipyype.testing import example_data
>>> aw = fsl.ApplyWarp()
>>> aw.inputs.in_file = example_data('structural.nii')
>>> aw.inputs.ref_file = example_data('mni.nii')
>>> aw.inputs.field_file = 'my_coefficients_filed.nii'
>>> res = aw.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         image to be warped
         flag: --in=%s, position: 0
ref_file: (an existing file name)
         reference image
         flag: --ref=%s, position: 1

[Optional]
abswarp: (a boolean)
         treat warp field as absolute: x' = w(x)
         flag: --abs
         mutually_exclusive: relwarp
args: (a unicode string)
      Additional parameters to the command
      flag: %s
datatype: ('char' or 'short' or 'int' or 'float' or 'double')
          Force output data type [char short int float double].
          flag: --datatype=%s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyype default value: {})
          Environment variables
field_file: (an existing file name)
            file containing warp field
            flag: --warp=%s
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
interp: ('nn' or 'trilinear' or 'sinc' or 'spline')
         interpolation method
         flag: --interp=%s, position: -2
```



```

mask_file: (an existing file name)
            filename for mask image (in reference space)
            flag: --mask=%s
out_file: (a file name)
           output filename
           flag: --out=%s, position: 2
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
             FSL output type
postmat: (an existing file name)
          filename for post-transform (affine matrix)
          flag: --postmat=%s
premat: (an existing file name)
         filename for pre-transform (affine matrix)
         flag: --premat=%s
relwarp: (a boolean)
          treat warp field as relative:  $x' = x + w(x)$ 
          flag: --rel, position: -1
          mutually_exclusive: abswarp
superlevel: ('a' or an integer (int or long))
             level of intermediary supersampling, a for 'automatic' or integer
             level. Default = 2
             flag: --superlevel=%s
supersample: (a boolean)
              intermediary supersampling of output, default is off
              flag: --super
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
          Warped output file

```

References:: None

ApplyXFM[Link to code](#)Wraps command **flirt**

Currently just a light wrapper around FLIRT, with no modifications

ApplyXFM is used to apply an existing transform to an image

Examples

```

>>> import nipy.interfaces.fsl as fsl
>>> from nipy.testing import example_data
>>> applyxfm = fsl.preprocess.ApplyXFM()
>>> applyxfm.inputs.in_file = example_data('structural.nii')
>>> applyxfm.inputs.in_matrix_file = example_data('trans.mat')
>>> applyxfm.inputs.out_file = 'newfile.nii'
>>> applyxfm.inputs.reference = example_data('mni.nii')
>>> applyxfm.inputs.apply_xfm = True
>>> result = applyxfm.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file
        flag: -in %s, position: 0
reference: (an existing file name)
        reference file
        flag: -ref %s, position: 1

[Optional]
angle_rep: ('quaternion' or 'euler')
        representation of rotation angles
        flag: -anglerep %s
apply_isoxfm: (a float)
        as applyxfm but forces isotropic resampling
        flag: -applyisoxfm %f
        mutually_exclusive: apply_xfm
apply_xfm: (a boolean, nipy default value: True)
        apply transformation supplied by in_matrix_file or uses_qform to use
        the affine matrix stored in the reference header
        flag: -applyxfm
args: (a unicode string)
        Additional parameters to the command
        flag: %s
bbrslope: (a float)
        value of bbr slope
        flag: -bbrslope %f
bbrtype: ('signed' or 'global_abs' or 'local_abs')
        type of bbr cost function: signed [default], global_abs, local_abs
        flag: -bbrtype %s
bgvalue: (a float)
        use specified background value for points outside FOV
        flag: -setbackground %f
bins: (an integer (int or long))
        number of histogram bins
        flag: -bins %d
coarse_search: (an integer (int or long))
        coarse search delta angle
        flag: -coarsesearch %d
cost: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
        'leastsq' or 'labeldiff' or 'bbr')
        cost function
        flag: -cost %s
cost_func: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
        'leastsq' or 'labeldiff' or 'bbr')
        cost function
        flag: -searchcost %s
datatype: ('char' or 'short' or 'int' or 'float' or 'double')
        force output data type
        flag: -datatype %s
display_init: (a boolean)
        display initial matrix
        flag: -displayinit
dof: (an integer (int or long))
        number of transform degrees of freedom
        flag: -dof %d
echospacing: (a float)
        value of EPI echo spacing - units of seconds
        flag: -echospacing %f

```

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
fieldmap: (a file name)
         fieldmap image in rads/s - must be already registered to the
         reference image
         flag: -fieldmap %s
fieldmapmask: (a file name)
         mask for fieldmap image
         flag: -fieldmapmask %s
fine_search: (an integer (int or long))
         fine search delta angle
         flag: -finesearch %d
force_scaling: (a boolean)
         force rescaling even for low-res images
         flag: -forcescaling
ignore_exception: (a boolean, nipy default value: False)
         Print an error message instead of throwing an exception in case the
         interface fails to run
in_matrix_file: (a file name)
         input 4x4 affine matrix
         flag: -init %s
in_weight: (an existing file name)
         File for input weighting volume
         flag: -inweight %s
interp: ('trilinear' or 'nearestneighbour' or 'sinc' or 'spline')
         final interpolation method used in reslicing
         flag: -interp %s
min_sampling: (a float)
         set minimum voxel dimension for sampling
         flag: -minsampling %f
no_clamp: (a boolean)
         do not use intensity clamping
         flag: -noclamp
no_resample: (a boolean)
         do not change input sampling
         flag: -noresample
no_resample_blur: (a boolean)
         do not use blurring on downsampling
         flag: -noresampblur
no_search: (a boolean)
         set all angular searches to ranges 0 to 0
         flag: -nosearch
out_file: (a file name)
         registered output file
         flag: -out %s, position: 2
out_log: (a file name)
         output log
         requires: save_log
out_matrix_file: (a file name)
         output affine matrix in 4x4 asciiii format
         flag: -omat %s, position: 3
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
             'NIFTI_PAIR_GZ')
             FSL output type
padding_size: (an integer (int or long))
             for applyxfm: interpolates outside image by size

```

```

    flag: -paddingsize %d
pedir: (an integer (int or long))
    phase encode direction of EPI - 1/2/3=x/y/z & -1/-2/-3=-x/-y/-z
    flag: -pedir %d
ref_weight: (an existing file name)
    File for reference weighting volume
    flag: -refweight %s
rigid2D: (a boolean)
    use 2D rigid body mode - ignores dof
    flag: -2D
save_log: (a boolean)
    save to log file
schedule: (an existing file name)
    replaces default schedule
    flag: -schedule %s
searchr_x: (a list of from 2 to 2 items which are an integer (int or
    long))
    search angles along x-axis, in degrees
    flag: -searchrx %s
searchr_y: (a list of from 2 to 2 items which are an integer (int or
    long))
    search angles along y-axis, in degrees
    flag: -searchry %s
searchr_z: (a list of from 2 to 2 items which are an integer (int or
    long))
    search angles along z-axis, in degrees
    flag: -searchrz %s
sinc_width: (an integer (int or long))
    full-width in voxels
    flag: -sincwidth %d
sinc_window: ('rectangular' or 'hanning' or 'blackman')
    sinc window
    flag: -sincwindow %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
uses_qform: (a boolean)
    initialize using sform or qform
    flag: -usesqform
verbose: (an integer (int or long))
    verbose mode, 0 is least
    flag: -verbose %d
wm_seg: (a file name)
    white matter segmentation volume needed by BBR cost function
    flag: -wmseg %s
wmcoords: (a file name)
    white matter boundary coordinates for BBR cost function
    flag: -wmcoords %s
wmnorms: (a file name)
    white matter boundary normals for BBR cost function
    flag: -wmnorms %s

```

Outputs:

```

out_file: (an existing file name)
    path/name of registered file (if generated)
out_log: (a file name)

```

```

    path/name of output log (if generated)
out_matrix_file: (an existing file name)
    path/name of calculated affine transform (if generated)

```

References:: None

BET

[Link to code](#)

Wraps command **bet**

Use FSL BET command for skull stripping.

For complete details, see the [BET Documentation](#).

Examples

```

>>> from nipy.interfaces import fsl
>>> btr = fsl.BET()
>>> btr.inputs.in_file = 'structural.nii'
>>> btr.inputs.frac = 0.7
>>> btr.inputs.out_file = 'brain_anat.nii'
>>> btr.cmdline
'bet structural.nii brain_anat.nii -f 0.70'
>>> res = btr.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input file to skull strip
    flag: %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
center: (a list of at most 3 items which are an integer (int or
    long))
    center of gravity in voxels
    flag: -c %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
frac: (a float)
    fractional intensity threshold
    flag: -f %.2f
functional: (a boolean)
    apply to 4D fMRI data
    flag: -F
    mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (a boolean)
    create binary mask image
    flag: -m
mesh: (a boolean)

```

```

        generate a vtk mesh brain surface
        flag: -e
no_output: (a boolean)
        Don't generate segmented output
        flag: -n
out_file: (a file name)
        name of output skull stripped image
        flag: %s, position: 1
outline: (a boolean)
        create surface outline image
        flag: -o
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
        FSL output type
padding: (a boolean)
        improve BET if FOV is very small in Z (by temporarily padding end
        slices)
        flag: -Z
        mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
radius: (an integer (int or long))
        head radius
        flag: -r %d
reduce_bias: (a boolean)
        bias field and neck cleanup
        flag: -B
        mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
remove_eyes: (a boolean)
        eye & optic nerve cleanup (can be useful in SIENA)
        flag: -S
        mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
robust: (a boolean)
        robust brain centre estimation (iterates BET several times)
        flag: -R
        mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
skull: (a boolean)
        create skull image
        flag: -s
surfaces: (a boolean)
        run bet2 and then betsurf to get additional skull and scalp surfaces
        (includes registrations)
        flag: -A
        mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
t2_guided: (a file name)
        as with creating surfaces, when also feeding in non-brain-extracted
        T2 (includes registrations)
        flag: -A2 %s
        mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
threshold: (a boolean)

```

```

        apply thresholding to segmented brain image and mask
        flag: -t
vertical_gradient: (a float)
        vertical gradient in fractional intensity threshold (-1, 1)
        flag: -g %.2f

```

Outputs:

```

inskull_mask_file: (a file name)
        path/name of inskull mask (if generated)
inskull_mesh_file: (a file name)
        path/name of inskull mesh outline (if generated)
mask_file: (a file name)
        path/name of binary brain mask (if generated)
meshfile: (a file name)
        path/name of vtk mesh file (if generated)
out_file: (a file name)
        path/name of skullstripped file (if generated)
outline_file: (a file name)
        path/name of outline file (if generated)
outskin_mask_file: (a file name)
        path/name of outskin mask (if generated)
outskin_mesh_file: (a file name)
        path/name of outskin mesh outline (if generated)
outskull_mask_file: (a file name)
        path/name of outskull mask (if generated)
outskull_mesh_file: (a file name)
        path/name of outskull mesh outline (if generated)
skull_mask_file: (a file name)
        path/name of skull mask (if generated)

```

References:: None

FAST[Link to code](#)Wraps command **fast**

Use FSL FAST for segmenting and bias correction.

For complete details, see the [FAST Documentation](#).**Examples**

```

>>> from nipy.interfaces import fsl
>>> fastr = fsl.FAST()
>>> fastr.inputs.in_files = 'structural.nii'
>>> fastr.inputs.out_basename = 'fast_'
>>> fastr.cmdline
'fast -o fast_ -S 1 structural.nii'
>>> out = fastr.run()

```

Inputs:

```

[Mandatory]
in_files: (a list of items which are an existing file name)
        image, or multi-channel set of images, to be segmented
        flag: %s, position: -1

[Optional]

```

```

args: (a unicode string)
    Additional parameters to the command
    flag: %s
bias_iters: (1 <= a long integer <= 10)
    number of main-loop iterations during bias-field removal
    flag: -I %d
bias_lowpass: (4 <= a long integer <= 40)
    bias field smoothing extent (FWHM) in mm
    flag: -l %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
hyper: (0.0 <= a floating point number <= 1.0)
    segmentation spatial smoothness
    flag: -H %.2f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
img_type: (1 or 2 or 3)
    int specifying type of image: (1 = T1, 2 = T2, 3 = PD)
    flag: -t %d
init_seg_smooth: (0.0001 <= a floating point number <= 0.1)
    initial segmentation spatial smoothness (during bias field
    estimation)
    flag: -f %.3f
init_transform: (an existing file name)
    <standard2input.mat> initialise using priors
    flag: -a %s
iters_afterbias: (1 <= a long integer <= 20)
    number of main-loop iterations after bias-field removal
    flag: -O %d
manual_seg: (an existing file name)
    Filename containing intensities
    flag: -s %s
mixel_smooth: (0.0 <= a floating point number <= 1.0)
    spatial smoothness for mixeltype
    flag: -R %.2f
no_bias: (a boolean)
    do not remove bias field
    flag: -N
no_pve: (a boolean)
    turn off PVE (partial volume estimation)
    flag: --nopve
number_classes: (1 <= a long integer <= 10)
    number of tissue-type classes
    flag: -n %d
other_priors: (a list of from 3 to 3 items which are a file name)
    alternative prior images
    flag: -A %s
out_basename: (a file name)
    base name of output files
    flag: -o %s
output_biascorrected: (a boolean)
    output restored image (bias-corrected image)
    flag: -B
output_biasfield: (a boolean)
    output estimated bias field

```



```

    flag: -b
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
    FSL output type
probability_maps: (a boolean)
    outputs individual probability maps
    flag: -p
segment_iters: (1 <= a long integer <= 50)
    number of segmentation-initialisation iterations
    flag: -W %d
segments: (a boolean)
    outputs a separate binary image for each tissue type
    flag: -g
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
use_priors: (a boolean)
    use priors throughout
    flag: -P
verbose: (a boolean)
    switch on diagnostic messages
    flag: -v

```

Outputs:

```

bias_field: (a list of items which are a file name)
mixeltype: (a file name)
    path/name of mixeltype volume file _mixeltype
partial_volume_files: (a list of items which are a file name)
partial_volume_map: (a file name)
    path/name of partial volume file _pveseg
probability_maps: (a list of items which are a file name)
restored_image: (a list of items which are a file name)
tissue_class_files: (a list of items which are a file name)
tissue_class_map: (an existing file name)
    path/name of binary segmented volume file one val for each class
    _seg

```

References:: None

FIRST[Link to code](#)Wraps command **run_first_all**

Use FSL's run_first_all command to segment subcortical volumes

<http://www.fmrib.ox.ac.uk/fsl/first/index.html>**Examples**

```

>>> from nipy.interfaces import fsl
>>> first = fsl.FIRST()
>>> first.inputs.in_file = 'structural.nii'
>>> first.inputs.out_file = 'segmented.nii'
>>> res = first.run()

```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input data file
    flag: -i %s, position: -2
out_file: (a file name, nipy default value: segmented)
    output data file
    flag: -o %s, position: -1

[Optional]
affine_file: (an existing file name)
    Affine matrix to use (e.g. img2std.mat) (does not re-run
    registration)
    flag: -a %s, position: 6
args: (a unicode string)
    Additional parameters to the command
    flag: %s
brain_extracted: (a boolean)
    Input structural image is already brain-extracted
    flag: -b, position: 2
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
list_of_specific_structures: (a list of at least 1 items which are a
    unicode string)
    Runs only on the specified structures (e.g. L_Hipp, R_HippL_Accu,
    R_Accu, L_Amyg, R_AmygL_Caud, R_Caud, L_Pall, R_PallL_Puta, R_Puta,
    L_Thal, R_Thal, BrStem)
    flag: -s %s, position: 5
method: ('auto' or 'fast' or 'none', nipy default value: auto)
    Method must be one of auto, fast, none, or it can be entered using
    the 'method_as_numerical_threshold' input
    flag: -m %s, position: 4
    mutually_exclusive: method_as_numerical_threshold
method_as_numerical_threshold: (a float)
    Specify a numerical threshold value or use the 'method' input to
    choose auto, fast, or none
    flag: -m %.4f, position: 4
no_cleanup: (a boolean)
    Input structural image is already brain-extracted
    flag: -d, position: 3
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Use verbose logging.
    flag: -v, position: 1
```

Outputs:

```

bvars: (a list of items which are an existing file name)
        bvars for each subcortical region
original_segmentations: (an existing file name)
        3D image file containing the segmented regions as integer values.
        Uses CMA labelling
segmentation_file: (an existing file name)
        4D image file containing a single volume per segmented region
vtk_surfaces: (a list of items which are an existing file name)
        VTK format meshes for each subcortical region

```

References:: None

FLIRT

[Link to code](#)

Wraps command **flirt**

Use FSL FLIRT for coregistration.

For complete details, see the [FLIRT Documentation](#).

To print out the command line help, use: `fsl.FLIRT().inputs_help()`

Examples

```

>>> from nipyype.interfaces import fsl
>>> from nipyype.testing import example_data
>>> flt = fsl.FLIRT(bins=640, cost_func='mutualinfo')
>>> flt.inputs.in_file = 'structural.nii'
>>> flt.inputs.reference = 'mni.nii'
>>> flt.inputs.output_type = "NIFTI_GZ"
>>> flt.cmdline
'flirt -in structural.nii -ref mni.nii -out structural_flirt.nii.gz -omat_
↳ structural_flirt.mat -bins 640 -searchcost mutualinfo'
>>> res = flt.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file
        flag: -in %s, position: 0
reference: (an existing file name)
        reference file
        flag: -ref %s, position: 1

[Optional]
angle_rep: ('quaternion' or 'euler')
        representation of rotation angles
        flag: -anglerep %s
apply_isoxfm: (a float)
        as applyxfm but forces isotropic resampling
        flag: -applyisoxfm %f
        mutually_exclusive: apply_xfm
apply_xfm: (a boolean)
        apply transformation supplied by in_matrix_file or uses_qform to use
        the affine matrix stored in the reference header
        flag: -applyxfm
args: (a unicode string)
        Additional parameters to the command
        flag: %s

```

```

bbrslope: (a float)
    value of bbr slope
    flag: -bbrslope %f
bbrtype: ('signed' or 'global_abs' or 'local_abs')
    type of bbr cost function: signed [default], global_abs, local_abs
    flag: -bbrtype %s
bgvalue: (a float)
    use specified background value for points outside FOV
    flag: -setbackground %f
bins: (an integer (int or long))
    number of histogram bins
    flag: -bins %d
coarse_search: (an integer (int or long))
    coarse search delta angle
    flag: -coarsesearch %d
cost: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
    'leastsq' or 'labeldiff' or 'bbr')
    cost function
    flag: -cost %s
cost_func: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
    'leastsq' or 'labeldiff' or 'bbr')
    cost function
    flag: -searchcost %s
datatype: ('char' or 'short' or 'int' or 'float' or 'double')
    force output data type
    flag: -datatype %s
display_init: (a boolean)
    display initial matrix
    flag: -displayinit
dof: (an integer (int or long))
    number of transform degrees of freedom
    flag: -dof %d
echospadding: (a float)
    value of EPI echo spacing - units of seconds
    flag: -echospadding %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fieldmap: (a file name)
    fieldmap image in rads/s - must be already registered to the
    reference image
    flag: -fieldmap %s
fieldmapmask: (a file name)
    mask for fieldmap image
    flag: -fieldmapmask %s
fine_search: (an integer (int or long))
    fine search delta angle
    flag: -finesearch %d
force_scaling: (a boolean)
    force rescaling even for low-res images
    flag: -forcescaling
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_matrix_file: (a file name)
    input 4x4 affine matrix
    flag: -init %s

```

```

in_weight: (an existing file name)
    File for input weighting volume
    flag: -inweight %s
interp: ('trilinear' or 'nearestneighbour' or 'sinc' or 'spline')
    final interpolation method used in reslicing
    flag: -interp %s
min_sampling: (a float)
    set minimum voxel dimension for sampling
    flag: -minsampling %f
no_clamp: (a boolean)
    do not use intensity clamping
    flag: -noclamp
no_resample: (a boolean)
    do not change input sampling
    flag: -noresample
no_resample_blur: (a boolean)
    do not use blurring on downsampling
    flag: -noresampblur
no_search: (a boolean)
    set all angular searches to ranges 0 to 0
    flag: -nosearch
out_file: (a file name)
    registered output file
    flag: -out %s, position: 2
out_log: (a file name)
    output log
    requires: save_log
out_matrix_file: (a file name)
    output affine matrix in 4x4 asciiii format
    flag: -omat %s, position: 3
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
padding_size: (an integer (int or long))
    for applyxfm: interpolates outside image by size
    flag: -paddingsize %d
pedir: (an integer (int or long))
    phase encode direction of EPI - 1/2/3=x/y/z & -1/-2/-3=-x/-y/-z
    flag: -pedir %d
ref_weight: (an existing file name)
    File for reference weighting volume
    flag: -refweight %s
rigid2D: (a boolean)
    use 2D rigid body mode - ignores dof
    flag: -2D
save_log: (a boolean)
    save to log file
schedule: (an existing file name)
    replaces default schedule
    flag: -schedule %s
searchr_x: (a list of from 2 to 2 items which are an integer (int or
    long))
    search angles along x-axis, in degrees
    flag: -searchrx %s
searchr_y: (a list of from 2 to 2 items which are an integer (int or
    long))
    search angles along y-axis, in degrees
    flag: -searchry %s

```

```

searchr_z: (a list of from 2 to 2 items which are an integer (int or
            long))
            search angles along z-axis, in degrees
            flag: -searchrz %s
sinc_width: (an integer (int or long))
            full-width in voxels
            flag: -sincwidth %d
sinc_window: ('rectangular' or 'hanning' or 'blackman')
            sinc window
            flag: -sincwindow %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
            Control terminal output: `stream` - displays to terminal immediately
            (default), `allatonce` - waits till command is finished to display
            output, `file` - writes output to file, `none` - output is ignored
uses_qform: (a boolean)
            initialize using sform or qform
            flag: -usesqform
verbose: (an integer (int or long))
            verbose mode, 0 is least
            flag: -verbose %d
wm_seg: (a file name)
            white matter segmentation volume needed by BBR cost function
            flag: -wmseg %s
wmcoords: (a file name)
            white matter boundary coordinates for BBR cost function
            flag: -wmcoords %s
wmnorms: (a file name)
            white matter boundary normals for BBR cost function
            flag: -wmnorms %s

```

Outputs:

```

out_file: (an existing file name)
            path/name of registered file (if generated)
out_log: (a file name)
            path/name of output log (if generated)
out_matrix_file: (an existing file name)
            path/name of calculated affine transform (if generated)

```

References:: None

FNIRT[Link to code](#)Wraps command **fnirt**

Use FSL FNIRT for non-linear registration.

For complete details, see the [FNIRT Documentation](#).**Examples**

```

>>> from nipy.interfaces import fsl
>>> from nipy.testing import example_data
>>> fnt = fsl.FNIRT(affine_file=example_data('trans.mat'))
>>> res = fnt.run(ref_file=example_data('mni.nii', in_file=example_data(
↪ 'structural.nii'))

```

T1 -> Mni153

```
>>> from nipy.interfaces import fsl
>>> fnirt_mprage = fsl.FNIRT()
>>> fnirt_mprage.inputs.in_fwhm = [8, 4, 2, 2]
>>> fnirt_mprage.inputs.subsampling_scheme = [4, 2, 1, 1]
```

Specify the resolution of the warps

```
>>> fnirt_mprage.inputs.warp_resolution = (6, 6, 6)
>>> res = fnirt_mprage.run(in_file='structural.nii', ref_file='mni.nii', warped_
↳ file='warped.nii', fieldcoeff_file='fieldcoeff.nii')
```

We can check the command line and confirm that it's what we expect.

```
>>> fnirt_mprage.cmdline
'fnirt --cout=fieldcoeff.nii --in=structural.nii --infwhm=8,4,2,2 --ref=mni.nii --
↳ subsamp=4,2,1,1 --warpres=6,6,6 --iout=warped.nii'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    name of input image
    flag: --in=%s
ref_file: (an existing file name)
    name of reference image
    flag: --ref=%s

[Optional]
affine_file: (an existing file name)
    name of file containing affine transform
    flag: --aff=%s
apply_inmask: (a list of items which are 0 or 1)
    list of iterations to use input mask on (1 to use, 0 to skip)
    flag: --applyinmask=%s
    mutually_exclusive: skip_inmask
apply_intensity_mapping: (a list of items which are 0 or 1)
    List of subsampling levels to apply intensity mapping for (0 to
    skip, 1 to apply)
    flag: --estint=%s
    mutually_exclusive: skip_intensity_mapping
apply_refmask: (a list of items which are 0 or 1)
    list of iterations to use reference mask on (1 to use, 0 to skip)
    flag: --applyrefmask=%s
    mutually_exclusive: skip_refmask
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bias_regularization_lambda: (a float)
    Weight of regularisation for bias-field, default 10000
    flag: --biaslambda=%f
biasfield_resolution: (a tuple of the form: (an integer (int or
    long), an integer (int or long), an integer (int or long)))
    Resolution (in mm) of bias-field modelling local intensities,
    default 50, 50, 50
    flag: --biasres=%d,%d,%d
config_file: ('T1_2_MNI152_2mm' or 'FA_2_FMRIB58_1mm' or an existing
    file name)
    Name of config file specifying command line arguments
    flag: --config=%s
derive_from_ref: (a boolean)
```

```
    If true, ref image is used to calculate derivatives. Default false
    flag: --refderiv
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
field_file: (a boolean or a file name)
    name of output file with field or true
    flag: --fout=%s
fieldcoeff_file: (a boolean or a file name)
    name of output file with field coefficients or true
    flag: --fout=%s
hessian_precision: ('double' or 'float')
    Precision for representing Hessian, double or float. Default double
    flag: --numprec=%s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_fwhm: (a list of items which are an integer (int or long))
    FWHM (in mm) of gaussian smoothing kernel for input volume, default
    [6, 4, 2, 2]
    flag: --infwhm=%s
in_intensitymap_file: (a list of from 1 to 2 items which are an
    existing file name)
    name of file/files containing initial intensity mapping usually
    generated by previous fnirt run
    flag: --intin=%s
inmask_file: (an existing file name)
    name of file with mask in input image space
    flag: --inmask=%s
inmask_val: (a float)
    Value to mask out in --in image. Default =0.0
    flag: --impinval=%f
intensity_mapping_model: ('none' or 'global_linear' or
    'global_non_linearlocal_linear' or 'global_non_linear_with_bias' or
    'local_non_linear')
    Model for intensity-mapping
    flag: --intmod=%s
intensity_mapping_order: (an integer (int or long))
    Order of poynomial for mapping intensities, default 5
    flag: --intorder=%d
inwarp_file: (an existing file name)
    name of file containing initial non-linear warps
    flag: --inwarp=%s
jacobian_file: (a boolean or a file name)
    name of file for writing out the Jacobian of the field (for
    diagnostic or VBM purposes)
    flag: --jout=%s
jacobian_range: (a tuple of the form: (a float, a float))
    Allowed range of Jacobian determinants, default 0.01, 100.0
    flag: --jacrange=%f,%f
log_file: (a file name)
    Name of log-file
    flag: --logout=%s
max_nonlin_iter: (a list of items which are an integer (int or long))
    Max # of non-linear iterations list, default [5, 5, 5, 5]
    flag: --miter=%s
modulatedref_file: (a boolean or a file name)
```



```

    name of file for writing out intensity modulated --ref (for
    diagnostic purposes)
    flag: --refout=%s
out_intensitymap_file: (a boolean or a file name)
    name of files for writing information pertaining to intensity
    mapping
    flag: --intout=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
ref_fwhm: (a list of items which are an integer (int or long))
    FWHM (in mm) of gaussian smoothing kernel for ref volume, default
    [4, 2, 0, 0]
    flag: --reffwhm=%s
refmask_file: (an existing file name)
    name of file with mask in reference space
    flag: --refmask=%s
refmask_val: (a float)
    Value to mask out in --ref image. Default =0.0
    flag: --imprefval=%f
regularization_lambda: (a list of items which are a float)
    Weight of regularisation, default depending on --ssqlambda and
    --regmod switches. See user documeta­tion.
    flag: --lambda=%s
regularization_model: ('membrane_energy' or 'bending_energy')
    Model for regularisation of warp-field [membrane_energy
    bending_energy], default bending_energy
    flag: --regmod=%s
skip_implicit_in_masking: (a boolean)
    skip implicit masking based on value in --in image. Default = 0
    flag: --impinm=0
skip_implicit_ref_masking: (a boolean)
    skip implicit masking based on value in --ref image. Default = 0
    flag: --imprefm=0
skip_inmask: (a boolean)
    skip specified inmask if set, default false
    flag: --applyinmask=0
    mutually_exclusive: apply_inmask
skip_intensity_mapping: (a boolean)
    Skip estimate intensity-mapping default false
    flag: --estint=0
    mutually_exclusive: apply_intensity_mapping
skip_lambda_ssq: (a boolean)
    If true, lambda is not weighted by current ssq, default false
    flag: --ssqlambda=0
skip_refmask: (a boolean)
    Skip specified refmask if set, default false
    flag: --applyrefmask=0
    mutually_exclusive: apply_refmask
spline_order: (an integer (int or long))
    Order of spline, 2->Qadratic spline, 3->Cubic spline. Default=3
    flag: --splineorder=%d
subsampling_scheme: (a list of items which are an integer (int or
    long))
    sub-sampling scheme, list, default [4, 2, 1, 1]
    flag: --subsamp=%s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately

```

```

        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
warp_resolution: (a tuple of the form: (an integer (int or long), an
        integer (int or long), an integer (int or long)))
        (approximate) resolution (in mm) of warp basis in x-, y- and
        z-direction, default 10, 10, 10
        flag: --warpres=%d,%d,%d
warped_file: (a file name)
        name of output image
        flag: --iout=%s

```

Outputs:

```

field_file: (a file name)
        file with warp field
fieldcoeff_file: (an existing file name)
        file with field coefficients
jacobian_file: (a file name)
        file containing Jacobian of the field
log_file: (a file name)
        Name of log-file
modulatedref_file: (a file name)
        file containing intensity modulated --ref
out_intensitymap_file: (a list of from 2 to 2 items which are a file
        name)
        files containing info pertaining to intensity mapping
warped_file: (an existing file name)
        warped image

```

References:: None

FUGUE[Link to code](#)Wraps command **fugue****FUGUE** is, most generally, a set of tools for EPI distortion correction.**Distortions may be corrected for**

1. improving registration with non-distorted images (e.g. structurals), or
2. dealing with motion-dependent changes.

FUGUE is designed to deal only with the first case - improving registration.**Examples**

Unwarping an input image (shift map is known)

```

>>> from nipy.interfaces.fsl.preprocess import FUGUE
>>> fugue = FUGUE()
>>> fugue.inputs.in_file = 'epi.nii'
>>> fugue.inputs.mask_file = 'epi_mask.nii'
>>> fugue.inputs.shift_in_file = 'vsm.nii' # Previously computed with fugue as well
>>> fugue.inputs.unwarp_direction = 'y'
>>> fugue.inputs.output_type = "NIFTI_GZ"
>>> fugue.cmdline
'fugue --in=epi.nii --mask=epi_mask.nii --loadshift=vsm.nii --unwarmdir=y --
  ↪unwarp=epi_unwarped.nii.gz'
>>> fugue.run()

```

Warping an input image (shift map is known)

```
>>> from nipy.interfaces.fsl.preprocess import FUGUE
>>> fugue = FUGUE()
>>> fugue.inputs.in_file = 'epi.nii'
>>> fugue.inputs.forward_warping = True
>>> fugue.inputs.mask_file = 'epi_mask.nii'
>>> fugue.inputs.shift_in_file = 'vsm.nii' # Previously computed with fugue as_
↳ well
>>> fugue.inputs.unwarp_direction = 'y'
>>> fugue.inputs.output_type = "NIFTI_GZ"
>>> fugue.cmdline
'fugue --in=epi.nii --mask=epi_mask.nii --loadshift=vsm.nii --unwarpcdir=y --
↳ warp=epi_warped.nii.gz'
>>> fugue.run()
```

Computing the vsm (unwrapped phase map is known)

```
>>> from nipy.interfaces.fsl.preprocess import FUGUE
>>> fugue = FUGUE()
>>> fugue.inputs.phasemap_in_file = 'epi_phasediff.nii'
>>> fugue.inputs.mask_file = 'epi_mask.nii'
>>> fugue.inputs.dwell_to_asym_ratio = (0.77e-3 * 3) / 2.46e-3
>>> fugue.inputs.unwarp_direction = 'y'
>>> fugue.inputs.save_shift = True
>>> fugue.inputs.output_type = "NIFTI_GZ"
>>> fugue.cmdline
'fugue --dwelltoasym=0.9390243902 --mask=epi_mask.nii --phasemap=epi_phasediff.
↳ nii --saveshift=epi_phasediff_vsm.nii.gz --unwarpcdir=y'
>>> fugue.run()
```

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
asym_se_time: (a float)
    set the fieldmap asymmetric spin echo time (sec)
    flag: --asym=%10f
despike_2dfilter: (a boolean)
    apply a 2D de-spiking filter
    flag: --despike
despike_threshold: (a float)
    specify the threshold for de-spiking (default=3.0)
    flag: --despikethreshold=%s
dwell_time: (a float)
    set the EPI dwell time per phase-encode line - same as echo spacing
    - (sec)
    flag: --dwell=%10f
dwell_to_asym_ratio: (a float)
    set the dwell to asym time ratio
    flag: --dwelltoasym=%10f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fmap_in_file: (an existing file name)
    filename for loading fieldmap (rad/s)
```

```
    flag: --loadfmap=%s
fmap_out_file: (a file name)
    filename for saving fieldmap (rad/s)
    flag: --savefmap=%s
forward_warping: (a boolean, nipy default value: False)
    apply forward warping instead of unwarping
fourier_order: (an integer (int or long))
    apply Fourier (sinusoidal) fitting of order N
    flag: --fourier=%d
icorr: (a boolean)
    apply intensity correction to unwarping (pixel shift method only)
    flag: --icorr
    requires: shift_in_file
icorr_only: (a boolean)
    apply intensity correction only
    flag: --icorronly
    requires: unwrapped_file
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file: (an existing file name)
    filename of input volume
    flag: --in=%s
mask_file: (an existing file name)
    filename for loading valid mask
    flag: --mask=%s
median_2dfilter: (a boolean)
    apply 2D median filtering
    flag: --median
no_extend: (a boolean)
    do not apply rigid-body extrapolation to the fieldmap
    flag: --noextend
no_gap_fill: (a boolean)
    do not apply gap-filling measure to the fieldmap
    flag: --nofill
nokspace: (a boolean)
    do not use k-space forward warping
    flag: --nokspace
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
pava: (a boolean)
    apply monotonic enforcement via PAVA
    flag: --pava
phase_conjugate: (a boolean)
    apply phase conjugate method of unwarping
    flag: --phaseconj
phasemap_in_file: (an existing file name)
    filename for input phase image
    flag: --phasemap=%s
poly_order: (an integer (int or long))
    apply polynomial fitting of order N
    flag: --poly=%d
save_fmap: (a boolean)
    write field map volume
    mutually_exclusive: save_unmasked_fmap
save_shift: (a boolean)
    write pixel shift volume
```

```

    mutually_exclusive: save_unmasked_shift
save_unmasked_fmap: (a boolean)
    saves the unmasked fieldmap when using --savefmap
    flag: --unmaskfmap
    mutually_exclusive: save_fmap
save_unmasked_shift: (a boolean)
    saves the unmasked shiftmap when using --saveshift
    flag: --unmaskshift
    mutually_exclusive: save_shift
shift_in_file: (an existing file name)
    filename for reading pixel shift volume
    flag: --loadshift=%s
shift_out_file: (a file name)
    filename for saving pixel shift volume
    flag: --saveshift=%s
smooth2d: (a float)
    apply 2D Gaussian smoothing of sigma N (in mm)
    flag: --smooth2=%.2f
smooth3d: (a float)
    apply 3D Gaussian smoothing of sigma N (in mm)
    flag: --smooth3=%.2f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
unwarp_direction: ('x' or 'y' or 'z' or 'x-' or 'y-' or 'z-')
    specifies direction of warping (default y)
    flag: --unwarpcdir=%s
unwarped_file: (a file name)
    apply unwarping and save as filename
    flag: --unwarp=%s
    mutually_exclusive: warped_file
    requires: in_file
warped_file: (a file name)
    apply forward warping and save as filename
    flag: --warp=%s
    mutually_exclusive: unwarped_file
    requires: in_file

```

Outputs:

```

fmap_out_file: (a file name)
    fieldmap file
shift_out_file: (a file name)
    voxel shift map file
unwarped_file: (a file name)
    unwarped file
warped_file: (a file name)
    forward warped file

```

References:: None

MCFLIRT[Link to code](#)Wraps command **mcflirt**

Use FSL MCFLIRT to do within-modality motion correction.

For complete details, see the [MCFLIRT Documentation](#).

Examples

```

>>> from nipy.interfaces import fsl
>>> mcflt = fsl.MCFLIRT()
>>> mcflt.inputs.in_file = 'functional.nii'
>>> mcflt.inputs.cost = 'mutualinfo'
>>> mcflt.inputs.out_file = 'moco.nii'
>>> mcflt.cmdline
'mcflirt -in functional.nii -cost mutualinfo -out moco.nii'
>>> res = mcflt.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         timeseries to motion-correct
         flag: -in %s, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
bins: (an integer (int or long))
      number of histogram bins
      flag: -bins %d
cost: ('mutualinfo' or 'woods' or 'corratio' or 'normcorr' or
       'normmi' or 'leastquares')
      cost function to optimize
      flag: -cost %s
dof: (an integer (int or long))
     degrees of freedom for the transformation
     flag: -dof %d
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
init: (an existing file name)
      initial transformation matrix
      flag: -init %s
interpolation: ('spline' or 'nn' or 'sinc')
               interpolation method for transformation
               flag: -%s_final
mean_vol: (a boolean)
          register to mean volume
          flag: -meanvol
out_file: (a file name)
          file to write
          flag: -out %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
             FSL output type
ref_file: (an existing file name)
          target image for motion correction
          flag: -reffile %s
ref_vol: (an integer (int or long))
         volume to align frames to

```

```

    flag: -refvol %d
rotation: (an integer (int or long))
    scaling factor for rotation tolerances
    flag: -rotation %d
save_mats: (a boolean)
    save transformation matrices
    flag: -mats
save_plots: (a boolean)
    save transformation parameters
    flag: -plots
save_rms: (a boolean)
    save rms displacement parameters
    flag: -rmsabs -rmsrel
scaling: (a float)
    scaling factor to use
    flag: -scaling %.2f
smooth: (a float)
    smoothing factor for the cost function
    flag: -smooth %.2f
stages: (an integer (int or long))
    stages (if 4, perform final search with sinc interpolation)
    flag: -stages %d
stats_imgs: (a boolean)
    produce variance and std. dev. images
    flag: -stats
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
use_contour: (a boolean)
    run search on contour images
    flag: -edge
use_gradient: (a boolean)
    run search on gradient images
    flag: -gdt

```

Outputs:

```

mat_file: (a list of items which are an existing file name)
    transformation matrices
mean_img: (an existing file name)
    mean timeseries image (if mean_vol=True)
out_file: (an existing file name)
    motion-corrected timeseries
par_file: (an existing file name)
    text-file with motion parameters
rms_files: (a list of items which are an existing file name)
    absolute and relative displacement parameters
std_img: (an existing file name)
    standard deviation image
variance_img: (an existing file name)
    variance image

```

References:: None

PRELUDE[Link to code](#)Wraps command **prelude**

Use FSL prelude to do phase unwrapping

Examples

Please insert examples for use of this command

Inputs:

```
[Mandatory]
complex_phase_file: (an existing file name)
    complex phase input volume
    flag: --complex=%s
    mutually_exclusive: magnitude_file, phase_file
magnitude_file: (an existing file name)
    file containing magnitude image
    flag: --abs=%s
    mutually_exclusive: complex_phase_file
phase_file: (an existing file name)
    raw phase file
    flag: --phase=%s
    mutually_exclusive: complex_phase_file

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
end: (an integer (int or long))
    final image number to process (default Inf)
    flag: --end=%d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
label_file: (a file name)
    saving the area labels output
    flag: --labels=%s
labelprocess2d: (a boolean)
    does label processing in 2D (slice at a time)
    flag: --labelslices
mask_file: (an existing file name)
    filename of mask input volume
    flag: --mask=%s
num_partitions: (an integer (int or long))
    number of phase partitions to use
    flag: --numphasesplit=%d
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
process2d: (a boolean)
    does all processing in 2D (slice at a time)
    flag: --slices
    mutually_exclusive: labelprocess2d
process3d: (a boolean)
    forces all processing to be full 3D
    flag: --force3D
    mutually_exclusive: labelprocess2d, process2d
rawphase_file: (a file name)
```



```

        saving the raw phase output
        flag: --rawphase=%s
removeramps: (a boolean)
        remove phase ramps during unwrapping
        flag: --removeramps
savemask_file: (a file name)
        saving the mask volume
        flag: --savemask=%s
start: (an integer (int or long))
        first image number to process (default 0)
        flag: --start=%d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
threshold: (a float)
        intensity threshold for masking
        flag: --thresh=%.10f
unwrapped_phase_file: (a file name)
        file containing unwrapepd phase
        flag: --unwrap=%s

```

Outputs:

```

unwrapped_phase_file: (an existing file name)
        unwrapped phase file

```

References:: None

SUSAN[Link to code](#)Wraps command **susan**

use FSL SUSAN to perform smoothing

For complete details, see the [SUSAN Documentation](#).**Examples**

```

>>> from nipy.interfaces import fsl
>>> from nipy.testing import example_data
>>> anatfile
anatomical.nii
>>> sus = fsl.SUSAN()
>>> sus.inputs.in_file = example_data('structural.nii')
>>> sus.inputs.brightness_threshold = 2000.0
>>> sus.inputs.fwhm = 8.0
>>> result = sus.run()

```

Inputs:

```

[Mandatory]
brightness_threshold: (a float)
        brightness threshold and should be greater than noise level and less
        than contrast of edges to be preserved.
        flag: %.10f, position: 2
fwhm: (a float)
        fwhm of smoothing, in mm, gets converted using sqrt(8*log(2))
        flag: %.10f, position: 3

```

```

in_file: (an existing file name)
         filename of input timeseries
         flag: %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
dimension: (3 or 2, nipy default value: 3)
            within-plane (2) or fully 3D (3)
            flag: %d, position: 4
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          output file name
          flag: %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
usans: (a list of at most 2 items which are a tuple of the form: (an
        existing file name, a float), nipy default value: [])
        determines whether the smoothing area (USAN) is to be found from
        secondary images (0, 1 or 2). A negative value for any brightness
        threshold will auto-set the threshold at 10% of the robust range
use_median: (1 or 0, nipy default value: 1)
             whether to use a local median filter in the cases where single-point
             noise is detected
             flag: %d, position: 5

```

Outputs:

```

smoothed_file: (an existing file name)
               smoothed output file

```

References:: None

SliceTimer[Link to code](#)Wraps command **slcetimer**use FSL **slcetimer** to perform slice timing correction.**Examples**

```

>>> from nipy.interfaces import fsl
>>> from nipy.testing import example_data
>>> st = fsl.SliceTimer()
>>> st.inputs.in_file = example_data('functional.nii')

```

```
>>> st.inputs.interleaved = True
>>> result = st.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        filename of input timeseries
        flag: --in=%s, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
custom_order: (an existing file name)
              filename of single-column custom interleave order file (first slice
              is referred to as 1 not 0)
              flag: --ocustom=%s
custom_timings: (an existing file name)
                slice timings, in fractions of TR, range 0:1 (default is 0.5 = no
                shift)
                flag: --tcustom=%s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
global_shift: (a float)
              shift in fraction of TR, range 0:1 (default is 0.5 = no shift)
              flag: --tglobal
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
index_dir: (a boolean)
            slice indexing from top to bottom
            flag: --down
interleaved: (a boolean)
             use interleaved acquisition
             flag: --odd
out_file: (a file name)
          filename of output timeseries
          flag: --out=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
slice_direction: (1 or 2 or 3)
                 direction of slice acquisition (x=1, y=2, z=3) - default is z
                 flag: --direction=%d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
time_repetition: (a float)
                  Specify TR of data - default is 3s
                  flag: --repeat=%f
```

Outputs:

```
slice_time_corrected_file: (an existing file name)
                           slice time corrected file
```

References:: None

interfaces.fsl.utils

AvScale

[Link to code](#)

Wraps command **avscale**

Use FSL avscale command to extract info from mat file output of FLIRT

Examples

```
>>> avscale = AvScale()
>>> avscale.inputs.mat_file = 'flirt.mat'
>>> res = avscale.run()
```

Inputs:

```
[Mandatory]

[Optional]
all_param: (a boolean)
            flag: --allparams
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipype default value: {})
          Environment variables
ignore_exception: (a boolean, nipype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mat_file: (an existing file name)
           mat file to read
           flag: %s, position: -2
ref_file: (an existing file name)
           reference file to get center of rotation
           flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
average_scaling: (a float)
                  Average Scaling
backward_half_transform: (a list of items which are a list of items
                          which are a float)
                          Backwards Half Transform
determinant: (a float)
              Determinant
forward_half_transform: (a list of items which are a list of items
                         which are a float)
                         Forward Half Transform
left_right_orientation_preserved: (a boolean)
                                  True if LR orientation preserved
```

```

rot_angles: (a list of items which are a float)
    rotation angles
rotation_translation_matrix: (a list of items which are a list of
    items which are a float)
    Rotation and Translation Matrix
scales: (a list of items which are a float)
    Scales (x,y,z)
skews: (a list of items which are a float)
    Skews
translations: (a list of items which are a float)
    translations

```

Complex

[Link to code](#)

Wraps command **fslcomplex**

fslcomplex is a tool for converting complex data

Examples

```

>>> cplx = Complex()
>>> cplx.inputs.complex_in_file = "complex.nii"
>>> cplx.real_polar = True
>>> res = cplx.run()

```

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
complex_cartesian: (a boolean)
    flag: -complex, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
complex_in_file: (an existing file name)
    flag: %s, position: 2
complex_in_file2: (an existing file name)
    flag: %s, position: 3
complex_merge: (a boolean)
    flag: -complexmerge, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge, start_vol, end_vol
complex_out_file: (a file name)
    flag: %s, position: -3
    mutually_exclusive: complex_out_file, magnitude_out_file,
        phase_out_file, real_out_file, imaginary_out_file, real_polar,
        real_cartesian
complex_polar: (a boolean)
    flag: -complexpolar, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
complex_split: (a boolean)
    flag: -complexsplit, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,

```

```

        complex_polar, complex_split, complex_merge
end_vol: (an integer (int or long))
        flag: %d, position: -1
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
imaginary_in_file: (an existing file name)
        flag: %s, position: 3
imaginary_out_file: (a file name)
        flag: %s, position: -3
        mutually_exclusive: complex_out_file, magnitude_out_file,
        phase_out_file, real_polar, complex_cartesian, complex_polar,
        complex_split, complex_merge
magnitude_in_file: (an existing file name)
        flag: %s, position: 2
magnitude_out_file: (a file name)
        flag: %s, position: -4
        mutually_exclusive: complex_out_file, real_out_file,
        imaginary_out_file, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_PAIR_GZ')
        FSL output type
phase_in_file: (an existing file name)
        flag: %s, position: 3
phase_out_file: (a file name)
        flag: %s, position: -3
        mutually_exclusive: complex_out_file, real_out_file,
        imaginary_out_file, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
real_cartesian: (a boolean)
        flag: -realcartesian, position: 1
        mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
real_in_file: (an existing file name)
        flag: %s, position: 2
real_out_file: (a file name)
        flag: %s, position: -4
        mutually_exclusive: complex_out_file, magnitude_out_file,
        phase_out_file, real_polar, complex_cartesian, complex_polar,
        complex_split, complex_merge
real_polar: (a boolean)
        flag: -realpolar, position: 1
        mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
start_vol: (an integer (int or long))
        flag: %d, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

complex_out_file: (a file name)
imaginary_out_file: (a file name)
magnitude_out_file: (a file name)
phase_out_file: (a file name)
real_out_file: (a file name)

```

References:: None

ConvertWarp

[Link to code](#)

Wraps command **convertwarp**

Use FSL [convertwarp](#) for combining multiple transforms into one.

Examples

```

>>> from nipy.interfaces.fsl import ConvertWarp
>>> warputils = ConvertWarp()
>>> warputils.inputs.warp1 = "warpfield.nii"
>>> warputils.inputs.reference = "T1.nii"
>>> warputils.inputs.relwarp = True
>>> warputils.inputs.output_type = "NIFTI_GZ"
>>> warputils.cmdline
'convertwarp --ref=T1.nii --rel --warp1=warpfield.nii --out=T1_concatwarp.nii.gz'
>>> res = warputils.run()

```

Inputs:

```

[Mandatory]
reference: (an existing file name)
    Name of a file in target space of the full transform.
    flag: --ref=%s, position: 1

[Optional]
abswarp: (a boolean)
    If set it indicates that the warps in --warp1 and --warp2 should be
    interpreted as absolute. I.e. the values in --warp1/2 are the
    coordinates in the next space, rather than displacements. This flag
    is ignored if --warp1/2 was created by fnirt, which always creates
    relative displacements.
    flag: --abs
    mutually_exclusive: relwarp
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cons_jacobian: (a boolean)
    Constrain the Jacobian of the warpfield to lie within specified
    min/max limits.
    flag: --constrainj
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
jacobian_max: (a float)
    Maximum acceptable Jacobian value for constraint (default 100.0)

```

```

    flag: --jmax=%f
jacobian_min: (a float)
    Minimum acceptable Jacobian value for constraint (default 0.01)
    flag: --jmin=%f
midmat: (an existing file name)
    Name of file containing mid-warp-affine transform
    flag: --midmat=%s
out_abswarp: (a boolean)
    If set it indicates that the warps in --out should be absolute, i.e.
    the values in --out are displacements from the coordinates in --ref.
    flag: --absout
    mutually_exclusive: out_relwarp
out_file: (a file name)
    Name of output file, containing warps that are the combination of
    all those given as arguments. The format of this will be a field-
    file (rather than spline coefficients) with any affine components
    included.
    flag: --out=%s, position: -1
out_relwarp: (a boolean)
    If set it indicates that the warps in --out should be relative, i.e.
    the values in --out are displacements from the coordinates in --ref.
    flag: --relout
    mutually_exclusive: out_abswarp
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
postmat: (an existing file name)
    Name of file containing an affine transform (applied last). It could
    e.g. be an affine transform that maps the MNI152-space into a better
    approximation to the Talairach-space (if indeed there is one).
    flag: --postmat=%s
premat: (an existing file name)
    filename for pre-transform (affine matrix)
    flag: --premat=%s
relwarp: (a boolean)
    If set it indicates that the warps in --warp1/2 should be
    interpreted as relative. I.e. the values in --warp1/2 are
    displacements from the coordinates in the next space.
    flag: --rel
    mutually_exclusive: abswarp
shift_direction: ('y-' or 'y' or 'x' or 'x-' or 'z' or 'z-')
    Indicates the direction that the distortions from --shiftmap goes.
    It depends on the direction and polarity of the phase-encoding in
    the EPI sequence.
    flag: --shiftdir=%s
    requires: shift_in_file
shift_in_file: (an existing file name)
    Name of file containing a "shiftmap", a non-linear transform with
    displacements only in one direction (applied first, before premat).
    This would typically be a fieldmap that has been pre-processed using
    fugue that maps a subjects functional (EPI) data onto an undistorted
    space (i.e. a space that corresponds to his/her true anatomy).
    flag: --shiftmap=%s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
warp1: (an existing file name)

```



```

        Name of file containing initial warp-fields/coefficients (follows
        premat). This could e.g. be a fnirt-transform from a subjects
        structural scan to an average of a group of subjects.
        flag: --warp1=%s
warp2: (an existing file name)
        Name of file containing secondary warp-fields/coefficients (after
        warp1/midmat but before postmat). This could e.g. be a fnirt-
        transform from the average of a group of subjects to some standard
        space (e.g. MNI152).
        flag: --warp2=%s

```

Outputs:

```

out_file: (an existing file name)
        Name of output file, containing the warp as field or coefficients.

```

References:: None

ConvertXFM[Link to code](#)Wraps command **convert_xfm**Use the FSL utility `convert_xfm` to modify FLIRT transformation matrices.**Examples**

```

>>> import nipy.interfaces.fsl as fsl
>>> invt = fsl.ConvertXFM()
>>> invt.inputs.in_file = "flirt.mat"
>>> invt.inputs.invert_xfm = True
>>> invt.inputs.out_file = 'flirt_inv.mat'
>>> invt.cmdline
'convert_xfm -omat flirt_inv.mat -inverse flirt.mat'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input transformation matrix
        flag: %s, position: -1

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
concat_xfm: (a boolean)
        write joint transformation of two input matrices
        flag: -concat, position: -3
        mutually_exclusive: invert_xfm, concat_xfm, fix_scale_skew
        requires: in_file2
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
fix_scale_skew: (a boolean)
        use secondary matrix to fix scale and skew
        flag: -fixscaleskew, position: -3
        mutually_exclusive: invert_xfm, concat_xfm, fix_scale_skew

```

```
requires: in_file2
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file2: (an existing file name)
    second input matrix (for use with fix_scale_skew or concat_xfm)
    flag: %s, position: -2
invert_xfm: (a boolean)
    invert input transformation
    flag: -inverse, position: -3
    mutually_exclusive: invert_xfm, concat_xfm, fix_scale_skew
out_file: (a file name)
    final transformation matrix
    flag: -omat %s, position: 1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    output transformation matrix
```

References:: None

CopyGeom

[Link to code](#)Wraps command **fslcpgeom**

Use fslcpgeom to copy the header geometry information to another image. Copy certain parts of the header information (image dimensions, voxel dimensions, voxel dimensions units string, image orientation/origin or qform/sform info) from one image to another. Note that only copies from Analyze to Analyze or Nifti to Nifti will work properly. Copying from different files will result in loss of information or potentially incorrect settings.

Inputs:

```
[Mandatory]
dest_file: (an existing file name)
    destination image
    flag: %s, position: 1
in_file: (an existing file name)
    source image
    flag: %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_dims: (a boolean)
    Do not copy image dimensions
    flag: -d, position: -1
```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    image with new geometry header

```

References:: None

ExtractROI[Link to code](#)Wraps command **fslroi**

Uses FSL Fslroi command to extract region of interest (ROI) from an image.

You can a) take a 3D ROI from a 3D data set (or if it is 4D, the same ROI is taken from each time point and a new 4D data set is created), b) extract just some time points from a 4D data set, or c) control time and space limits to the ROI. Note that the arguments are minimum index and size (not maximum index). So to extract voxels 10 to 12 inclusive you would specify 10 and 3 (not 10 and 12).

Examples

```

>>> from nipy.interfaces.fsl import ExtractROI
>>> from nipy.testing import anatfile
>>> fslroi = ExtractROI(in_file=anatfile, roi_file='bar.nii', t_min=0,
...                    t_size=1)
>>> fslroi.cmdline == 'fslroi %s bar.nii 0 1' % anatfile
True

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input file
    flag: %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
crop_list: (a list of items which are a tuple of the form: (an
    integer (int or long), an integer (int or long)))
    list of two tuples specifying crop options
    flag: %s, position: 2
    mutually_exclusive: x_min, x_size, y_min, y_size, z_min, z_size,
    t_min, t_size
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
roi_file: (a file name)
    output file
    flag: %s, position: 1
t_min: (an integer (int or long))
    flag: %d, position: 8
t_size: (an integer (int or long))
    flag: %d, position: 9
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
x_min: (an integer (int or long))
    flag: %d, position: 2
x_size: (an integer (int or long))
    flag: %d, position: 3
y_min: (an integer (int or long))
    flag: %d, position: 4
y_size: (an integer (int or long))
    flag: %d, position: 5
z_min: (an integer (int or long))
    flag: %d, position: 6
z_size: (an integer (int or long))
    flag: %d, position: 7

```

Outputs:

```
roi_file: (an existing file name)
```

References:: None

FilterRegressor[Link to code](#)**Wraps command `fsl_regfilt`**

Data de-noising by regressing out part of a design matrix

Uses simple OLS regression on 4D images

Inputs:

```

[Mandatory]
design_file: (an existing file name)
    name of the matrix with time courses (e.g. GLM design or MELODIC
    mixing matrix)
    flag: -d %s, position: 3
filter_all: (a boolean)
    use all columns in the design file in denoising
    flag: -f '%s', position: 4
    mutually_exclusive: filter_columns
filter_columns: (a list of items which are an integer (int or long))
    (1-based) column indices to filter out of the data
    flag: -f '%s', position: 4
    mutually_exclusive: filter_all
in_file: (an existing file name)
    input file name (4D image)

```

```

        flag: -i %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    mask image file name
    flag: -m %s
out_file: (a file name)
    output file name for the filtered data
    flag: -o %s, position: 2
out_vnscales: (a boolean)
    output scaling factors for variance normalization
    flag: --out_vnscales
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
var_norm: (a boolean)
    perform variance-normalization on data
    flag: --vn

```

Outputs:

```

out_file: (an existing file name)
    output file name for the filtered data

```

References:: None

ImageMaths[Link to code](#)Wraps command **fslmaths**Use FSL fslmaths command to allow mathematical manipulation of images [FSL info](#)**Examples**

```

>>> from nipype.interfaces import fsl
>>> from nipype.testing import anatfile
>>> maths = fsl.ImageMaths(in_file=anatfile, op_string= '-add 5',
...                        out_file='foo_maths.nii')
>>> maths.cmdline == 'fslmaths %s -add 5 foo_maths.nii' % anatfile
True

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        flag: %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
in_file2: (an existing file name)
          flag: %s, position: 3
op_string: (a unicode string)
           string defining the operation, i. e. -add
           flag: %s, position: 2
out_data_type: ('char' or 'short' or 'int' or 'float' or 'double' or
               'input')
               output datatype, one of (char, short, int, float, double, input)
               flag: -odt %s, position: 5
out_file: (a file name)
          flag: %s, position: 4
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
             'NIFTI_PAIR_GZ')
             FSL output type
suffix: (a unicode string)
        out_file suffix
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
                 (default), `allatonce` - waits till command is finished to display
                 output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```
out_file: (an existing file name)
```

References:: None

ImageMeants[Link to code](#)Wraps command **fslmeants**

Use fslmeants for printing the average timeseries (intensities) to the screen (or saves to a file). The average is taken over all voxels in the mask (or all voxels in the image if no mask is specified)

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file for computing the average timeseries
        flag: -i %s, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s

```

```

eig: (a boolean)
    calculate Eigenvariate(s) instead of mean (output will have 0 mean)
    flag: --eig
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    input 3D mask
    flag: -m %s
nobin: (a boolean)
    do not binarise the mask for calculation of Eigenvariates
    flag: --no_bin
order: (an integer (int or long), nipy default value: 1)
    select number of Eigenvariates
    flag: --order=%d
out_file: (a file name)
    name of output text matrix
    flag: -o %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
show_all: (a boolean)
    show all voxel time series (within mask) instead of averaging
    flag: --showall
spatial_coord: (a list of items which are an integer (int or long))
    <x y z> requested spatial coordinate (instead of mask)
    flag: -c %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transpose: (a boolean)
    output results in transpose format (one row per voxel/mean)
    flag: --transpose
use_mm: (a boolean)
    use mm instead of voxel coordinates (for -c option)
    flag: --usemm

```

Outputs:

```

out_file: (an existing file name)
    path/name of output text matrix

```

References:: None

ImageStats[Link to code](#)Wraps command **fsstats**Use FSL fsstats command to calculate stats from images [FSL info](#)

Examples

```
>>> from nipyte.interfaces.fsl import ImageStats
>>> from nipyte.testing import funcfile
>>> stats = ImageStats(in_file=funcfile, op_string= '-M')
>>> stats.cmdline == 'fslstats %s -M'%funcfile
True
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input file to generate stats of
    flag: %s, position: 2
op_string: (a unicode string)
    string defining the operation, options are applied in order, e.g. -M
    -l 10 -M will report the non-zero mean, apply a threshold and then
    report the new nonzero mean
    flag: %s, position: 3

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_file: (an existing file name)
    mask file used for option -k %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
split_4d: (a boolean)
    give a separate output line for each 3D volume of a 4D timeseries
    flag: -t, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_stat: (any value)
    stats output
```

References:: None

InvWarp

[Link to code](#)

Wraps command **invwarp**

Use FSL Invwarp to invert a FNIRT warp

Examples

```
>>> from nipy.interfaces.fsl import InvWarp
>>> invwarp = InvWarp()
>>> invwarp.inputs.warp = "struct2mni.nii"
>>> invwarp.inputs.reference = "anatomical.nii"
>>> invwarp.inputs.output_type = "NIFTI_GZ"
>>> invwarp.cmdline
'invwarp --out=struct2mni_inverse.nii.gz --ref=anatomical.nii --warp=struct2mni.
↳nii'
>>> res = invwarp.run()
```

Inputs:

```
[Mandatory]
reference: (an existing file name)
    Name of a file in target space. Note that the target space is now
    different from the target space that was used to create the --warp
    file. It would typically be the file that was specified with the
    --in argument when running fnirt.
    flag: --ref=%s
warp: (an existing file name)
    Name of file containing warp-coefficients/fields. This would
    typically be the output from the --cout switch of fnirt (but can
    also use fields, like the output from --fout).
    flag: --warp=%s

[Optional]
absolute: (a boolean)
    If set it indicates that the warps in --warp should be interpreted
    as absolute, provided that it is not created by fnirt (which always
    uses relative warps). If set it also indicates that the output --out
    should be absolute.
    flag: --abs
    mutually_exclusive: relative
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inverse_warp: (a file name)
    Name of output file, containing warps that are the "reverse" of
    those in --warp. This will be a field-file (rather than a file of
    spline coefficients), and it will have any affine component included
    as part of the displacements.
    flag: --out=%s
jacobian_max: (a float)
    Maximum acceptable Jacobian value for constraint (default 100.0)
    flag: --jmax=%f
jacobian_min: (a float)
    Minimum acceptable Jacobian value for constraint (default 0.01)
    flag: --jmin=%f
niter: (an integer (int or long))
    Determines how many iterations of the gradient-descent search that
```

```

        should be run.
        flag: --niter=%d
noconstraint: (a boolean)
    Do not apply Jacobian constraint
    flag: --noconstraint
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
    FSL output type
regularise: (a float)
    Regularization strength (default=1.0).
    flag: --regularise=%f
relative: (a boolean)
    If set it indicates that the warps in --warp should be interpreted
    as relative. I.e. the values in --warp are displacements from the
    coordinates in the --ref space. If set it also indicates that the
    output --out should be relative.
    flag: --rel
    mutually_exclusive: absolute
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

inverse_warp: (an existing file name)
    Name of output file, containing warps that are the "reverse" of
    those in --warp.

```

References:: None

Merge[Link to code](#)Wraps command **fslmerge**Use **fslmerge** to concatenate images

Images can be concatenated across time, x, y, or z dimensions. Across the time (t) dimension the TR is set by default to 1 sec.

Note: to set the TR to a different value, specify 't' for dimension and specify the TR value in seconds for the tr input. The dimension will be automatically updated to 'tr'.

Examples

```

>>> from nipy.interfaces.fsl import Merge
>>> merger = Merge()
>>> merger.inputs.in_files = ['functional2.nii', 'functional3.nii']
>>> merger.inputs.dimension = 't'
>>> merger.inputs.output_type = 'NIFTI_GZ'
>>> merger.cmdline
'fslmerge -t functional2_merged.nii.gz functional2.nii functional3.nii'
>>> merger.inputs.tr = 2.25
>>> merger.cmdline
'fslmerge -tr functional2_merged.nii.gz functional2.nii functional3.nii 2.25'

```

Inputs:

```

[Mandatory]
dimension: ('t' or 'x' or 'y' or 'z' or 'a')

```

```

        dimension along which to merge, optionally set tr input when
        dimension is t
        flag: -%s, position: 0
in_files: (a list of items which are an existing file name)
        flag: %s, position: 2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipype default value: {})
        Environment variables
ignore_exception: (a boolean, nipype default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
merged_file: (a file name)
        flag: %s, position: 1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_PAIR_GZ')
        FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
tr: (a float)
        use to specify TR in seconds (default is 1.00 sec), overrides
        dimension and sets it to tr
        flag: %.2f, position: -1

```

Outputs:

```
merged_file: (an existing file name)
```

References:: None

MotionOutliers[Link to code](#)Wraps command **fsl_motion_outliers**

Use FSL `fsl_motion_outliers` <http://fsl.fmrib.ox.ac.uk/fsl/fslwiki/FSLMotionOutliers> to find outliers in time-series (4d) data. Examples ~~~~~>>> from nipype.interfaces.fsl import MotionOutliers>>> mo = MotionOutliers()>>> mo.inputs.in_file = "epi.nii">>> mo.cmdline # doctest: +ELLIPSIS +ALLOW_UNICODE 'fsl_motion_outliers -i epi.nii -o epi_outliers.txt -p epi_metrics.png -s epi_metrics.txt'>>> res = mo.run() # doctest: +SKIP

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        unfiltered 4D image
        flag: -i %s

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
dummy: (an integer (int or long))

```

```

    number of dummy scans to delete (before running anything and
    creating EVs)
    flag: --dummy=%d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (an existing file name)
    mask image for calculating metric
    flag: -m %s
metric: ('refrms' or 'dvars' or 'refmse' or 'fd' or 'fdrms')
    metrics: refrms - RMS intensity difference to reference volume as
    metric [default metric], refmse - Mean Square Error version of
    refrms (used in original version of fsl_motion_outliers), dvars -
    DVARS, fd - frame displacement, fdrms - FD with RMS matrix
    calculation
    flag: --%s
no_motion_correction: (a boolean)
    do not run motion correction (assumed already done)
    flag: --nomoco
out_file: (a file name)
    output outlier file name
    flag: -o %s
out_metric_plot: (a file name)
    output metric values plot (DVARS etc.) file name
    flag: -p %s
out_metric_values: (a file name)
    output metric values (DVARS etc.) file name
    flag: -s %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (a float)
    specify absolute threshold value (otherwise use box-plot cutoff =
    P75 + 1.5*IQR)
    flag: --thresh=%g

```

Outputs:

```

out_file: (an existing file name)
out_metric_plot: (an existing file name)
out_metric_values: (an existing file name)

```

References:: None

Overlay[Link to code](#)Wraps command **overlay****Use FSL's overlay command to combine background and statistical images** into one volume

Examples

```
>>> from nipy.interfaces import fsl
>>> combine = fsl.Overlay()
>>> combine.inputs.background_image = 'mean_func.nii.gz'
>>> combine.inputs.auto_thresh_bg = True
>>> combine.inputs.stat_image = 'zstat1.nii.gz'
>>> combine.inputs.stat_thresh = (3.5, 10)
>>> combine.inputs.show_negative_stats = True
>>> res = combine.run()
```

Inputs:

```
[Mandatory]
auto_thresh_bg: (a boolean)
    automatically threshold the background image
    flag: -a, position: 5
    mutually_exclusive: auto_thresh_bg, full_bg_range, bg_thresh
background_image: (an existing file name)
    image to use as background
    flag: %s, position: 4
bg_thresh: (a tuple of the form: (a float, a float))
    min and max values for background intensity
    flag: %.3f %.3f, position: 5
    mutually_exclusive: auto_thresh_bg, full_bg_range, bg_thresh
full_bg_range: (a boolean)
    use full range of background image
    flag: -A, position: 5
    mutually_exclusive: auto_thresh_bg, full_bg_range, bg_thresh
stat_image: (an existing file name)
    statistical image to overlay in color
    flag: %s, position: 6
stat_thresh: (a tuple of the form: (a float, a float))
    min and max values for the statistical overlay
    flag: %.2f %.2f, position: 7

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    combined image volume
    flag: %s, position: -1
out_type: ('float' or 'int', nipy default value: float)
    write output with float or int
    flag: %s, position: 2
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
show_negative_stats: (a boolean)
    display negative statistics in overlay
    flag: %s, position: 8
```

```

        mutually_exclusive: stat_image2
stat_image2: (an existing file name)
        second statistical image to overlay in color
        flag: %s, position: 9
        mutually_exclusive: show_negative_stats
stat_thresh2: (a tuple of the form: (a float, a float))
        min and max values for second statistical overlay
        flag: %.2f %.2f, position: 10
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
transparency: (a boolean, nipy default value: True)
        make overlay colors semi-transparent
        flag: %s, position: 1
use_checkerboard: (a boolean)
        use checkerboard mask for overlay
        flag: -c, position: 3

```

Outputs:

```

out_file: (an existing file name)
        combined image volume

```

References:: None

PlotMotionParams[Link to code](#)Wraps command **fsl_tsp**

Use fsl_tsp to plot the estimated motion parameters from a realignment program.

Examples

```

>>> import nipy.interfaces.fsl as fsl
>>> plotter = fsl.PlotMotionParams()
>>> plotter.inputs.in_file = 'functional.par'
>>> plotter.inputs.in_source = 'fsl'
>>> plotter.inputs.plot_type = 'rotations'
>>> res = plotter.run()

```

Notes

The ‘in_source’ attribute determines the order of columns that are expected in the source file. FSL prints motion parameters in the order rotations, translations, while SPM prints them in the opposite order. This interface should be able to plot timecourses of motion parameters generated from other sources as long as they fall under one of these two patterns. For more flexibility, see the `fsl.PlotTimeSeries` interface.

Inputs:

```

[Mandatory]
in_file: (an existing file name or a list of items which are an
        existing file name)
        file with motion parameters
        flag: %s, position: 1
in_source: ('spm' or 'fsl')
        which program generated the motion parameter file - fsl, spm
plot_type: ('rotations' or 'translations' or 'displacement')

```

```

        which motion type to plot - rotations, translations, displacement
        flag: %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    image to write
    flag: -o %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
plot_size: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    plot image height and width
    flag: %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    image to write

```

References:: None

PlotTimeSeries[Link to code](#)Wraps command **fsl_tsplo**t

Use fsl_tsplo to create images of time course plots.

Examples

```

>>> import nipype.interfaces.fsl as fsl
>>> plotter = fsl.PlotTimeSeries()
>>> plotter.inputs.in_file = 'functional.par'
>>> plotter.inputs.title = 'Functional timeseries'
>>> plotter.inputs.labels = ['run1', 'run2']
>>> plotter.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name or a list of items which are an
    existing file name)
    file or list of files with columns of timecourse information
    flag: %s, position: 1

```

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
labels: (a unicode string or a list of items which are a unicode
    string)
    label or list of labels
    flag: %s
legend_file: (an existing file name)
    legend file
    flag: --legend=%s
out_file: (a file name)
    image to write
    flag: -o %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
plot_finish: (an integer (int or long))
    final column from in-file to plot
    flag: --finish=%d
    mutually_exclusive: plot_range
plot_range: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    first and last columns from the in-file to plot
    flag: %s
    mutually_exclusive: plot_start, plot_finish
plot_size: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    plot image height and width
    flag: %s
plot_start: (an integer (int or long))
    first column from in-file to plot
    flag: --start=%d
    mutually_exclusive: plot_range
sci_notation: (a boolean)
    switch on scientific notation
    flag: --sci
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
title: (a unicode string)
    plot title
    flag: %s
x_precision: (an integer (int or long))
    precision of x-axis labels
    flag: --precision=%d
x_units: (an integer (int or long), nipy default value: 1)
    scaling units for x-axis (between 1 and length of in file)
    flag: -u %d
```



```

y_max: (a float)
        maximum y value
        flag: --ymax=%.2f
        mutually_exclusive: y_range
y_min: (a float)
        minumum y value
        flag: --ymin=%.2f
        mutually_exclusive: y_range
y_range: (a tuple of the form: (a float, a float))
        min and max y axis values
        flag: %s
        mutually_exclusive: y_min, y_max

```

Outputs:

```

out_file: (an existing file name)
          image to write

```

References:: None

PowerSpectrum[Link to code](#)Wraps command **fslpspec**

Use FSL PowerSpectrum command for power spectrum estimation.

Examples

```

>>> from nipy.interfaces import fsl
>>> pspec = fsl.PowerSpectrum()
>>> pspec.inputs.in_file = 'functional.nii'
>>> res = pspec.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input 4D file to estimate the power spectrum
        flag: %s, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
out_file: (a file name)
          name of output 4D file for power spectrum
          flag: %s, position: 1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
             'NIFTI_PAIR_GZ')
             FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')

```

```
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          path/name of the output 4D power spectrum file
```

References:: None

Reorient2Std

[Link to code](#)

Wraps command **fslreorient2std**

fslreorient2std is a tool for reorienting the image to match the approximate orientation of the standard template images (MNI152).

Examples

```
>>> reorient = Reorient2Std()
>>> reorient.inputs.in_file = "functional.nii"
>>> res = reorient.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        flag: %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          flag: %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
```

References:: None

RobustFOV

[Link to code](#)

Wraps command **robustfov**

Automatically crops an image removing lower head and neck.

Interface is stable 5.0.0 to 5.0.9, but default brainsize changed from 150mm to 170mm.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input filename
        flag: -i %s, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
brainsize: (an integer (int or long))
           size of brain in z-dimension (default 170mm/150mm)
           flag: -b %d
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_roi: (a file name)
         ROI volume output name
         flag: -r %s
out_transform: (a file name)
               Transformation matrix in_file to out_roi output name
               flag: -m %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_roi: (an existing file name)
         ROI volume output name
out_transform: (an existing file name)
               Transformation matrix in_file to out_roi output name

```

References:: None

SigLoss

[Link to code](#)

Wraps command **sigloss**

Estimates signal loss from a field map (in rad/s)

Examples

```
>>> sigloss = SigLoss()
>>> sigloss.inputs.in_file = "phase.nii"
>>> sigloss.inputs.echo_time = 0.03
>>> res = sigloss.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        b0 fieldmap file
        flag: -i %s

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
echo_time: (a float)
           echo time in seconds
           flag: --te=%f
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
mask_file: (an existing file name)
           brain mask file
           flag: -m %s
out_file: (a file name)
          output signal loss estimate file
          flag: -s %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
              FSL output type
slice_direction: ('x' or 'y' or 'z')
                 slicing direction
                 flag: -d %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          signal loss estimate file
```

References:: None

Slicer

[Link to code](#)

Wraps command **slicer**

Use FSL's slicer command to output a png image from a volume.

Examples

```
>>> from nipyre.interfaces import fsl
>>> from nipyre.testing import example_data
>>> slice = fsl.Slicer()
>>> slice.inputs.in_file = example_data('functional.nii')
>>> slice.inputs.all_axial = True
>>> slice.inputs.image_width = 750
>>> res = slice.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input volume
        flag: %s, position: 1

[Optional]
all_axial: (a boolean)
           output all axial slices into one picture
           flag: -A, position: 10
           mutually_exclusive: single_slice, middle_slices, all_axial,
                               sample_axial
           requires: image_width
args: (a unicode string)
      Additional parameters to the command
      flag: %s
colour_map: (an existing file name)
            use different colour map from that stored in nifti header
            flag: -l %s, position: 4
dither_edges: (a boolean)
              produce semi-transparent (dithered) edges
              flag: -t, position: 7
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyre default value: {})
          Environment variables
ignore_exception: (a boolean, nipyre default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
image_edges: (an existing file name)
             volume to display edge overlay for (useful for checking registration)
             flag: %s, position: 2
image_width: (an integer (int or long))
             max picture width
             flag: %d, position: -2
intensity_range: (a tuple of the form: (a float, a float))
                 min and max intensities to display
                 flag: -i %.3f %.3f, position: 5
label_slices: (a boolean, nipyre default value: True)
              display slice number
              flag: -L, position: 3
middle_slices: (a boolean)
               output picture of mid-sagittal, axial, and coronal slices
               flag: -a, position: 10
               mutually_exclusive: single_slice, middle_slices, all_axial,
                                   sample_axial
nearest_neighbour: (a boolean)
                  use nearest neighbor interpolation for output
```

```

    flag: -n, position: 8
out_file: (a file name)
    picture to write
    flag: %s, position: -1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_PAIR_GZ')
    FSL output type
sample_axial: (an integer (int or long))
    output every n axial slices into one picture
    flag: -S %d, position: 10
    mutually_exclusive: single_slice, middle_slices, all_axial,
    sample_axial
    requires: image_width
scaling: (a float)
    image scale
    flag: -s %f, position: 0
show_orientation: (a boolean, nipy default value: True)
    label left-right orientation
    flag: %s, position: 9
single_slice: ('x' or 'y' or 'z')
    output picture of single slice in the x, y, or z plane
    flag: -%s, position: 10
    mutually_exclusive: single_slice, middle_slices, all_axial,
    sample_axial
    requires: slice_number
slice_number: (an integer (int or long))
    slice number to save in picture
    flag: -%d, position: 11
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold_edges: (a float)
    use threshold for edges
    flag: -e %.3f, position: 6

```

Outputs:

```

out_file: (an existing file name)
    picture to write

```

References:: None

Smooth[Link to code](#)Wraps command **fslmaths**

Use fslmaths to smooth the image

Examples

Setting the kernel width using sigma:

```

>>> sm = Smooth()
>>> sm.inputs.output_type = 'NIFTI_GZ'
>>> sm.inputs.in_file = 'functional2.nii'
>>> sm.inputs.sigma = 8.0
>>> sm.cmdline
'fslmaths functional2.nii -kernel gauss 8.000 -fmean functional2_smooth.nii.gz'

```

Setting the kernel width using fwhm:

```
>>> sm = Smooth()
>>> sm.inputs.output_type = 'NIFTI_GZ'
>>> sm.inputs.in_file = 'functional2.nii'
>>> sm.inputs.fwhm = 8.0
>>> sm.cmdline
'fslmaths functional2.nii -kernel gauss 3.397 -fmean functional2_smooth.nii.gz'
```

One of sigma or fwhm must be set:

```
>>> from nipyre.interfaces.fsl import Smooth
>>> sm = Smooth()
>>> sm.inputs.output_type = 'NIFTI_GZ'
>>> sm.inputs.in_file = 'functional2.nii'
>>> sm.cmdline
Traceback (most recent call last):
  ~~~
ValueError: Smooth requires a value for one of the inputs ...
```

Inputs:

```
[Mandatory]
fwhm: (a float)
    gaussian kernel fwhm, will be converted to sigma in mm (not voxels)
    flag: -kernel gauss %.03f -fmean, position: 1
    mutually_exclusive: sigma
in_file: (an existing file name)
    flag: %s, position: 0
sigma: (a float)
    gaussian kernel sigma in mm (not voxels)
    flag: -kernel gauss %.03f -fmean, position: 1
    mutually_exclusive: fwhm

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
smoothed_file: (a file name)
    flag: %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
smoothed_file: (an existing file name)
```

References:: None

Split

[Link to code](#)

Wraps command **fslsplit**

Uses FSL Fslsplit command to separate a volume into images in time, x, y or z dimension.

Inputs:

```
[Mandatory]
dimension: ('t' or 'x' or 'y' or 'z')
    dimension along which the file will be split
    flag: -%s, position: 2
in_file: (an existing file name)
    input filename
    flag: %s, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_base_name: (a unicode string)
    outputs prefix
    flag: %s, position: 1
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_files: (a list of items which are an existing file name)
```

References:: None

SwapDimensions

[Link to code](#)

Wraps command **fslswapdim**

Use fslswapdim to alter the orientation of an image.

This interface accepts a three-tuple corresponding to the new orientation. You may either provide dimension ids in the form of (-)x, (-)y, or (-)z, or nifti-syle dimension codes (RL, LR, AP, PA, IS, SI).

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input image
    flag: %s, position: 1
new_dims: (a tuple of the form: ('x' or '-x' or 'y' or '-y' or 'z' or
    '-z' or 'RL' or 'LR' or 'AP' or 'PA' or 'IS' or 'SI', 'x' or '-x'
    or 'y' or '-y' or 'z' or '-z' or 'RL' or 'LR' or 'AP' or 'PA' or
```



```

        'IS' or 'SI', 'x' or '-x' or 'y' or '-y' or 'z' or '-z' or 'RL' or
        'LR' or 'AP' or 'PA' or 'IS' or 'SI'))
    3-tuple of new dimension order
    flag: %s %s %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    image to write
    flag: %s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    image with new dimensions

```

References:: None

WarpPoints

[Link to code](#)

Wraps command **img2imgcoord**

Use FSL **img2imgcoord** to transform point sets. Accepts plain text files and vtk files.

Note: transformation of TrackVis trk files is not yet implemented

Examples

```

>>> from nipy.interfaces.fsl import WarpPoints
>>> warppoints = WarpPoints()
>>> warppoints.inputs.in_coords = 'surf.txt'
>>> warppoints.inputs.src_file = 'epi.nii'
>>> warppoints.inputs.dest_file = 'T1.nii'
>>> warppoints.inputs.warp_file = 'warpfield.nii'
>>> warppoints.inputs.coord_mm = True
>>> warppoints.cmdline
'img2imgcoord -mm -dest T1.nii -src epi.nii -warp warpfield.nii surf.txt'
>>> res = warppoints.run()

```

Inputs:

```
[Mandatory]
dest_file: (an existing file name)
    filename of destination image
    flag: -dest %s
in_coords: (an existing file name)
    filename of file containing coordinates
    flag: %s, position: -1
src_file: (an existing file name)
    filename of source image
    flag: -src %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
coord_mm: (a boolean)
    all coordinates in mm
    flag: -mm
    mutually_exclusive: coord_vox
coord_vox: (a boolean)
    all coordinates in voxels - default
    flag: -vox
    mutually_exclusive: coord_mm
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    output file name
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
warp_file: (an existing file name)
    filename of warpfield (e.g. intermediate2dest_warp.nii.gz)
    flag: -warp %s
    mutually_exclusive: xfm_file
xfm_file: (an existing file name)
    filename of affine transform (e.g. source2dest.mat)
    flag: -xfm %s
    mutually_exclusive: warp_file
```

Outputs:

```
out_file: (an existing file name)
    Name of output file, containing the warp as field or coefficients.
```

WarpPointsToStd

[Link to code](#)

Wraps command **img2stdcoord**

Use FSL **img2stdcoord** to transform point sets to standard space coordinates. Accepts plain text files and vtk files.

Note: transformation of TrackVis trk files is not yet implemented

Examples

```
>>> from nipy.interfaces.fsl import WarpPointsToStd
>>> warppoints = WarpPointsToStd()
>>> warppoints.inputs.in_coords = 'surf.txt'
>>> warppoints.inputs.img_file = 'T1.nii'
>>> warppoints.inputs.std_file = 'mni.nii'
>>> warppoints.inputs.warp_file = 'warpfield.nii'
>>> warppoints.inputs.coord_mm = True
>>> warppoints.cmdline
'img2stdcoord -mm -img T1.nii -std mni.nii -warp warpfield.nii surf.txt'
>>> res = warppoints.run()
```

Inputs:

```
[Mandatory]
img_file: (an existing file name)
    filename of input image
    flag: -img %s
in_coords: (an existing file name)
    filename of file containing coordinates
    flag: %s, position: -1
std_file: (an existing file name)
    filename of destination image
    flag: -std %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
coord_mm: (a boolean)
    all coordinates in mm
    flag: -mm
    mutually_exclusive: coord_vox
coord_vox: (a boolean)
    all coordinates in voxels - default
    flag: -vox
    mutually_exclusive: coord_mm
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    output file name
premat_file: (an existing file name)
    filename of pre-warp affine transform (e.g.
    example_func2highres.mat)
    flag: -premat %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
```

```

        output, `file` - writes output to file, `none` - output is ignored
warp_file: (an existing file name)
        filename of warpfield (e.g. intermediate2dest_warp.nii.gz)
        flag: -warp %s
        mutually_exclusive: xfm_file
xfm_file: (an existing file name)
        filename of affine transform (e.g. source2dest.mat)
        flag: -xfm %s
        mutually_exclusive: warp_file

```

Outputs:

```

out_file: (an existing file name)
        Name of output file, containing the warp as field or coefficients.

```

WarpUtils

[Link to code](#)Wraps command **fnirtfileutils**Use FSL **fnirtfileutils** to convert field->coefficients, coefficients->field, coefficients->other_coefficients etc

Examples

```

>>> from nipy.interfaces.fsl import WarpUtils
>>> warputils = WarpUtils()
>>> warputils.inputs.in_file = "warpfield.nii"
>>> warputils.inputs.reference = "T1.nii"
>>> warputils.inputs.out_format = 'spline'
>>> warputils.inputs.warp_resolution = (10,10,10)
>>> warputils.inputs.output_type = "NIFTI_GZ"
>>> warputils.cmdline
'fnirtfileutils --in=warpfield.nii --outformat=spline --ref=T1.nii --warpres=10.
↪0000,10.0000,10.0000 --out=warpfield_coeffs.nii.gz'
>>> res = invwarp.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        Name of file containing warp-coefficients/fields. This would
        typically be the output from the --cout switch of fnirt (but can
        also use fields, like the output from --fout).
        flag: --in=%s
reference: (an existing file name)
        Name of a file in target space. Note that the target space is now
        different from the target space that was used to create the --warp
        file. It would typically be the file that was specified with the
        --in argument when running fnirt.
        flag: --ref=%s
write_jacobian: (a boolean, nipy default value: False)
        Switch on --jac flag with automatically generated filename

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value

```

```

    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
knot_space: (a tuple of the form: (an integer (int or long), an
    integer (int or long), an integer (int or long)))
    Alternative (to --warpres) specification of the resolution of the
    output spline-field.
    flag: --knotspace=%d,%d,%d
out_file: (a file name)
    Name of output file. The format of the output depends on what other
    parameters are set. The default format is a (4D) field-file. If the
    --outformat is set to spline the format will be a (4D) file of
    spline coefficients.
    flag: --out=%s, position: -1
out_format: ('spline' or 'field')
    Specifies the output format. If set to field (default) the output
    will be a (4D) field-file. If set to spline the format will be a
    (4D) file of spline coefficients.
    flag: --outformat=%s
out_jacobian: (a file name)
    Specifies that a (3D) file of Jacobian determinants corresponding to
    --in should be produced and written to filename.
    flag: --jac=%s
output_type: ('NIFTI_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_PAIR_GZ')
    FSL output type
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
warp_resolution: (a tuple of the form: (a float, a float, a float))
    Specifies the resolution/knot-spacing of the splines pertaining to
    the coefficients in the --out file. This parameter is only relevant
    if --outformat is set to spline. It should be noted that if the --in
    file has a higher resolution, the resulting coefficients will
    pertain to the closest (in a least-squares sense) file in the space
    of fields with the --warpres resolution. It should also be noted
    that the resolution will always be an integer multiple of the voxel
    size.
    flag: --warpres=%0.4f,%0.4f,%0.4f
with_affine: (a boolean)
    Specifies that the affine transform (i.e. that which was specified
    for the --aff parameter in fnirt) should be included as
    displacements in the --out file. That can be useful for interfacing
    with software that cannot decode FSL/fnirt coefficient-files (where
    the affine transform is stored separately from the displacements).
    flag: --withaff

```

Outputs:

```

out_file: (a file name)
    Name of output file, containing the warp as field or coefficients.
out_jacobian: (a file name)
    Name of output file, containing the map of the determinant of the
    Jacobian

```

References:: None

interfaces.minc.base

aggregate_filename()

[Link to code](#)

Try to work out a sensible name given a set of files that have been combined in some way (e.g. averaged). If we can't work out a sensible prefix, we use the first filename in the list.

Examples

```
>>> from nipy.interfaces.minc.base import aggregate_filename
>>> f = aggregate_filename(['/tmp/fool.mnc', '/tmp/foo2.mnc', '/tmp/foo3.mnc'],
↳ 'averaged')
>>> os.path.split(f)[1] # This has a full path, so just check the filename.
'foo_averaged.mnc'
```

```
>>> f = aggregate_filename(['/tmp/fool.mnc', '/tmp/blah1.mnc'], 'averaged')
>>> os.path.split(f)[1] # This has a full path, so just check the filename.
'fool_averaged.mnc'
```

interfaces.minc.minc

Average

[Link to code](#)

Wraps command **mincaverage**

Average a number of MINC files.

Examples

```
>>> from nipy.interfaces.minc import Average
>>> from nipy.interfaces.minc.testdata import nonempty_minc_data
```

```
>>> files = [nonempty_minc_data(i) for i in range(3)]
>>> average = Average(input_files=files, output_file='/tmp/tmp.mnc')
>>> average.run()
```

Inputs:

```
[Mandatory]
filelist: (a file name)
    Specify the name of a file containing input file names.
    flag: -filelist %s
    mutually_exclusive: input_files, filelist
input_files: (a list of items which are a file name)
    input file(s)
    flag: %s, position: -2
    mutually_exclusive: input_files, filelist

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
avgdim: (a unicode string)
    Specify a dimension along which we wish to average.
    flag: -avgdim %s
binarize: (a boolean)
    Binarize the volume by looking for values in a given range.
    flag: -binarize
binrange: (a tuple of the form: (a float, a float))
    Specify a range for binarization. Default value: 1.79769e+308
    -1.79769e+308.
    flag: -binrange %s %s
binvalue: (a float)
    Specify a target value (+/- 0.5) for binarization. Default value:
    -1.79769e+308
    flag: -binvalue %s
check_dimensions: (a boolean)
    Check that dimension info matches across files (default).
    flag: -check_dimensions
    mutually_exclusive: check_dimensions, no_check_dimensions
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
copy_header: (a boolean)
    Copy all of the header from the first file (default for one file).
    flag: -copy_header
    mutually_exclusive: copy_header, no_copy_header
debug: (a boolean)
    Print out debugging messages.
    flag: -debug
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
format_byte: (a boolean)
    Write out byte data.
    flag: -byte
    mutually_exclusive: format_filetype, format_byte, format_short,
    format_int, format_long, format_float, format_double,
    format_signed, format_unsigned
```



```

format_double: (a boolean)
    Write out double-precision floating-point data.
    flag: -double
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_filetype: (a boolean)
    Use data type of first file (default).
    flag: -filetype
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_float: (a boolean)
    Write out single-precision floating-point data.
    flag: -float
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_int: (a boolean)
    Write out 32-bit integer data.
    flag: -int
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_long: (a boolean)
    Superseded by -int.
    flag: -long
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_short: (a boolean)
    Write out short integer data.
    flag: -short
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_signed: (a boolean)
    Write signed integer data.
    flag: -signed
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_unsigned: (a boolean)
    Write unsigned integer data (default).
    flag: -unsigned
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
max_buffer_size_in_kb: (a long integer >= 0)
    Specify the maximum size of the internal buffers (in kbytes).
    flag: -max_buffer_size_in_kb %d
no_check_dimensions: (a boolean)
    Do not check dimension info.
    flag: -nocheck_dimensions
    mutually_exclusive: check_dimensions, no_check_dimensions

```

```
no_copy_header: (a boolean)
    Do not copy all of the header from the first file (default for many
    files)).
    flag: -nocopy_header
    mutually_exclusive: copy_header, no_copy_header
nonnormalize: (a boolean)
    Do not normalize data sets (default).
    flag: -nonnormalize
    mutually_exclusive: normalize, nonnormalize
normalize: (a boolean)
    Normalize data sets for mean intensity.
    flag: -normalize
    mutually_exclusive: normalize, nonnormalize
output_file: (a file name)
    output file
    flag: %s, position: -1
quiet: (a boolean)
    Do not print out log messages.
    flag: -quiet
    mutually_exclusive: verbose, quiet
sdfile: (a file name)
    Specify an output sd file (default=None).
    flag: -sdfile %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
two: (a boolean)
    Create a MINC 2 output file.
    flag: -2
verbose: (a boolean)
    Print out log messages (default).
    flag: -verbose
    mutually_exclusive: verbose, quiet
voxel_range: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    Valid range for output data.
    flag: -range %d %d
weights: (a list of items which are a unicode string)
    Specify weights for averaging ("
```

Outputs:

```
output_file: (an existing file name)
    output file
```

BBox[Link to code](#)Wraps command **mincbbox**

Determine a bounding box of image.

Examples

```
>>> from nipyte.interfaces.minc import BBox
>>> from nipyte.interfaces.minc.testdata import nonempty_minc_data
```

```
>>> file0 = nonempty_minc_data(0)
>>> bbox = BBox(input_file=file0)
>>> bbox.run()
```

Inputs:

```
[Mandatory]
input_file: (an existing file name)
    input file
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
format_minccrop: (a boolean)
    Output format for minccrop: (-xlim x1 x2 -ylim y1 y2 -zlim z1 z2
    flag: -minccrop
format_minresample: (a boolean)
    Output format for minresample: (-step x y z -start x y z -nelements
    x y z
    flag: -minresample
format_minreshape: (a boolean)
    Output format for minreshape: (-start x,y,z -count dx,dy,dz
    flag: -minreshape
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
one_line: (a boolean)
    Output on one line (default): start_x y z width_x y z
    flag: -one_line
    mutually_exclusive: one_line, two_lines
out_file: (a file name)
    flag: > %s, position: -1
output_file: (a file name)
    output file containing bounding box corners
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (an integer (int or long))
    VIO_Real value threshold for bounding box. Default value: 0.
    flag: -threshold
two_lines: (a boolean)
    Output on two lines: start_x y z
    width_x y z
    flag: -two_lines
    mutually_exclusive: one_line, two_lines
```

Outputs:

```
output_file: (an existing file name)
             output file containing bounding box corners
```

Beast

[Link to code](#)

Wraps command **mincbeast**

Extract brain image using BEaST (Brain Extraction using non-local Segmentation Technique).

Examples

```
>>> from nipy.interfaces.minc import Beast
>>> from nipy.interfaces.minc.testdata import nonempty_minc_data
```

```
>>> file0 = nonempty_minc_data(0)
>>> beast = Beast(input_file=file0)
>>> beast .run()
```

Inputs:

```
[Mandatory]
input_file: (a file name)
            input file
            flag: %s, position: -2
library_dir: (a directory name)
            library directory
            flag: %s, position: -3

[Optional]
abspath: (a boolean, nipy default value: True)
         File paths in the library are absolute (default is relative to
         library root).
         flag: -abspath
args: (a unicode string)
      Additional parameters to the command
      flag: %s
clobber: (a boolean, nipy default value: True)
         Overwrite existing file.
         flag: -clobber
confidence_level_alpha: (a float)
                        Specify confidence level Alpha. Default value: 0.5
                        flag: -alpha %s
configuration_file: (a file name)
                   Specify configuration file.
                   flag: -configuration %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
fill_holes: (a boolean)
            Fill holes in the binary output.
            flag: -fill
flip_images: (a boolean)
             Flip images around the mid-sagittal plane to increase patch count.
             flag: -flip
ignore_exception: (a boolean, nipy default value: False)
```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
load_moments: (a boolean)
    Do not calculate moments instead use precalculatedlibrary moments.
    (for optimization purposes)
    flag: -load_moments
median_filter: (a boolean)
    Apply a median filter on the probability map.
    flag: -median
nlm_filter: (a boolean)
    Apply an NLM filter on the probability map (experimental).
    flag: -nlm_filter
number_selected_images: (an integer (int or long))
    Specify number of selected images. Default value: 20
    flag: -selection_num %s
output_file: (a file name)
    output file
    flag: %s, position: -1
patch_size: (an integer (int or long))
    Specify patch size for single scale approach. Default value: 1.
    flag: -patch_size %s
probability_map: (a boolean)
    Output the probability map instead of crisp mask.
    flag: -probability
same_resolution: (a boolean)
    Output final mask with the same resolution as input file.
    flag: -same_resolution
search_area: (an integer (int or long))
    Specify size of search area for single scale approach. Default
    value: 2.
    flag: -search_area %s
smoothness_factor_beta: (a float)
    Specify smoothness factor Beta. Default value: 0.25
    flag: -beta %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold_patch_selection: (a float)
    Specify threshold for patch selection. Default value: 0.95
    flag: -threshold %s
voxel_size: (an integer (int or long))
    Specify voxel size for calculations (4, 2, or 1).Default value: 4.
    Assumes no multiscale. Use configurationfile for multiscale.
    flag: -voxel_size %s

```

Outputs:

```

output_file: (an existing file name)
    output mask file

```

BestLinReg

[Link to code](#)

Wraps command **bestlinreg**

Hierachial linear fitting between two files.

The bestlinreg script is part of the EZminc package:

<https://github.com/BIC-MNI/EZminc/blob/master/scripts/bestlinreg.pl>

Examples

```
>>> from nipyype.interfaces.minc import BestLinReg
>>> from nipyype.interfaces.minc.testdata import nonempty_minc_data
```

```
>>> input_file = nonempty_minc_data(0)
>>> target_file = nonempty_minc_data(1)
>>> linreg = BestLinReg(source=input_file, target=target_file)
>>> linreg.run()
```

Inputs:

```
[Mandatory]
source: (an existing file name)
        source Minc file
        flag: %s, position: -4
target: (an existing file name)
        target Minc file
        flag: %s, position: -3

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
clobber: (a boolean, nipyype default value: True)
          Overwrite existing file.
          flag: -clobber
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipyype default value: {})
          Environment variables
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
output_mnc: (a file name)
             output mnc file
             flag: %s, position: -1
output_xfm: (a file name)
             output xfm file
             flag: %s, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
          Print out log messages. Default: False.
          flag: -verbose
```

Outputs:

```
output_mnc: (an existing file name)
            output mnc file
output_xfm: (an existing file name)
            output xfm file
```

BigAverage

[Link to code](#)

Wraps command mincbigaverage

Average 1000's of MINC files in linear time.

mincbigaverage is designed to discretise the problem of averaging either a large number of input files or averaging a smaller number of large files. (>1GB each). There is also some code included to perform “robust” averaging in which only the most common features are kept via down-weighting outliers beyond a standard deviation.

One advantage of mincbigaverage is that it avoids issues around the number of possible open files in HDF/netCDF. In short if you have more than 100 files open at once while averaging things will slow down significantly.

mincbigaverage does this via a iterative approach to averaging files and is a direct drop in replacement for mincaverage. That said not all the arguments of mincaverage are supported in mincbigaverage but they should be.

This tool is part of the minc-widgets package:

<https://github.com/BIC-MNI/minc-widgets/blob/master/mincbigaverage/mincbigaverage>

Examples

```
>>> from nipy.interfaces.minc import BigAverage
>>> from nipy.interfaces.minc.testdata import nonempty_minc_data
```

```
>>> files = [nonempty_minc_data(i) for i in range(3)]
>>> average = BigAverage(input_files=files, output_float=True, robust=True)
>>> average.run()
```

Inputs:

```
[Mandatory]
input_files: (a list of items which are a file name)
    input file(s)
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: --clobber
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_file: (a file name)
    output file
    flag: %s, position: -1
output_float: (a boolean)
    Output files with float precision.
    flag: --float
robust: (a boolean)
    Perform robust averaging, features that are outside 1
    standarddeviation from the mean are downweighted. Works well for
    noisydata with artifacts. see the --tmpdir option if you have a large
    number of input files.
    flag: -robust
```

```
sd_file: (a file name)
    Place standard deviation image in specified file.
    flag: --sdfile %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tmpdir: (a directory name)
    temporary files directory
    flag: -tmpdir %s
verbose: (a boolean)
    Print out log messages. Default: False.
    flag: --verbose
```

Outputs:

```
output_file: (an existing file name)
    output file
sd_file: (an existing file name)
    standard deviation image
```

Blob

[Link to code](#)

Wraps command **mincblob**

Calculate blobs from minc deformation grids.

Examples

```
>>> from nipy.interfaces.minc import Blob
>>> from nipy.interfaces.minc.testdata import minc2Dfile
```

```
>>> blob = Blob(input_file=minc2Dfile, output_file='/tmp/tmp.mnc', trace=True)
>>> blob.run()
```

Inputs:

```
[Mandatory]
input_file: (an existing file name)
    input file to blob
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
determinant: (a boolean)
    compute the determinant (exact growth and shrinkage) -- SLOW
    flag: -determinant
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
magnitude: (a boolean)
```



```

        compute the magnitude of the displacement vector
        flag: -magnitude
output_file: (a file name)
        output file
        flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
trace: (a boolean)
        compute the trace (approximate growth and shrinkage) -- FAST
        flag: -trace
translation: (a boolean)
        compute translation (structure displacement)
        flag: -translation

```

Outputs:

```

output_file: (an existing file name)
        output file

```

Blur[Link to code](#)Wraps command **mincblur**

Convolve an input volume with a Gaussian blurring kernel of user-defined width. Optionally, the first partial derivatives and the gradient magnitude volume can be calculated.

Examples

```

>>> from nipy.interfaces.minc import Blur
>>> from nipy.interfaces.minc.testdata import minc3Dfile

```

(1) Blur an input volume with a 6mm fwhm isotropic Gaussian blurring kernel:

```

>>> blur = Blur(input_file=minc3Dfile, fwhm=6, output_file_base='/tmp/out_6')
>>> blur.run()

```

mincblur will create /tmp/out_6_blur.mnc.

2. Calculate the blurred and gradient magnitude data:

```

>>> blur = Blur(input_file=minc3Dfile, fwhm=6, gradient=True, output_file_base='/
↳tmp/out_6')
>>> blur.run()

```

will create /tmp/out_6_blur.mnc and /tmp/out_6_dxyz.mnc.

(3) Calculate the blurred data, the partial derivative volumes and the gradient magnitude for the same data:

```

>>> blur = Blur(input_file=minc3Dfile, fwhm=6, partial=True, output_file_base='/
↳tmp/out_6')
>>> blur.run()

```

will create /tmp/out_6_blur.mnc, /tmp/out_6_dx.mnc, /tmp/out_6_dy.mnc, /tmp/out_6_dz.mnc and /tmp/out_6_dxyz.mnc.

Inputs:

```

[Mandatory]
fwhm: (a float)

```

```

    Full-width-half-maximum of gaussian kernel. Default value: 0.
    flag: -fwhm %s
    mutually_exclusive: fwhm, fwhm3d, standard_dev
fwhm3d: (a tuple of the form: (a float, a float, a float))
    Full-width-half-maximum of gaussian kernel.Default value:
    -1.79769e+308 -1.79769e+308 -1.79769e+308.
    flag: -3dfwhm %s %s %s
    mutually_exclusive: fwhm, fwhm3d, standard_dev
input_file: (an existing file name)
    input file
    flag: %s, position: -2
standard_dev: (a float)
    Standard deviation of gaussian kernel. Default value: 0.
    flag: -standarddev %s
    mutually_exclusive: fwhm, fwhm3d, standard_dev

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
dimensions: (1 or 2 or 3)
    Number of dimensions to blur (either 1,2 or 3). Default value: 3.
    flag: -dimensions %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gaussian: (a boolean)
    Use a gaussian smoothing kernel (default).
    flag: -gaussian
    mutually_exclusive: gaussian, rect
gradient: (a boolean)
    Create the gradient magnitude volume as well.
    flag: -gradient
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
no_apodize: (a boolean)
    Do not apodize the data before blurring.
    flag: -no_apodize
output_file_base: (a file name)
    output file base
    flag: %s, position: -1
partial: (a boolean)
    Create the partial derivative and gradient magnitude volumes as
    well.
    flag: -partial
rect: (a boolean)
    Use a rect (box) smoothing kernel.
    flag: -rect
    mutually_exclusive: gaussian, rect
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

gradient_dxyz: (a file name)
    Gradient dxyz.
output_file: (an existing file name)
    Blurred output file.
partial_dx: (a file name)
    Partial gradient dx.
partial_dxyz: (a file name)
    Partial gradient dxyz.
partial_dy: (a file name)
    Partial gradient dy.
partial_dz: (a file name)
    Partial gradient dz.

```

Calc[Link to code](#)Wraps command **minccalc**

Compute an expression using MINC files as input.

Examples

```

>>> from nipy.interfaces.minc import Calc
>>> from nipy.interfaces.minc.testdata import nonempty_minc_data

```

```

>>> file0 = nonempty_minc_data(0)
>>> file1 = nonempty_minc_data(1)
>>> calc = Calc(input_files=[file0, file1], output_file='/tmp/calc.mnc',
↳expression='A[0] + A[1]') # add files together
>>> calc.run()

```

Inputs:

```

[Mandatory]
expfile: (a file name)
    Name of file containing expression.
    flag: -expfile %s
    mutually_exclusive: expression, expfile
expression: (a unicode string)
    Expression to use in calculations.
    flag: -expression '%s'
    mutually_exclusive: expression, expfile
filelist: (a file name)
    Specify the name of a file containing input file names.
    flag: -filelist %s
    mutually_exclusive: input_files, filelist
input_files: (a list of items which are a file name)
    input file(s) for calculation
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
check_dimensions: (a boolean)
    Check that files have matching dimensions (default).

```

```
    flag: -check_dimensions
    mutually_exclusive: check_dimensions, no_check_dimensions
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
copy_header: (a boolean)
    Copy all of the header from the first file.
    flag: -copy_header
    mutually_exclusive: copy_header, no_copy_header
debug: (a boolean)
    Print out debugging messages.
    flag: -debug
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
eval_width: (an integer (int or long))
    Number of voxels to evaluate simultaneously.
    flag: -eval_width %s
format_byte: (a boolean)
    Write out byte data.
    flag: -byte
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_double: (a boolean)
    Write out double-precision floating-point data.
    flag: -double
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_filetype: (a boolean)
    Use data type of first file (default).
    flag: -filetype
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_float: (a boolean)
    Write out single-precision floating-point data.
    flag: -float
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_int: (a boolean)
    Write out 32-bit integer data.
    flag: -int
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_long: (a boolean)
    Superseded by -int.
    flag: -long
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_short: (a boolean)
    Write out short integer data.
    flag: -short
```

```

    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_signed: (a boolean)
    Write signed integer data.
    flag: -signed
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_unsigned: (a boolean)
    Write unsigned integer data (default).
    flag: -unsigned
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
ignore_nan: (a boolean)
    Ignore invalid data (NaN) for accumulations.
    flag: -ignore_nan
max_buffer_size_in_kb: (a long integer >= 0)
    Specify the maximum size of the internal buffers (in kbytes).
    flag: -max_buffer_size_in_kb %d
no_check_dimensions: (a boolean)
    Do not check that files have matching dimensions.
    flag: -nocheck_dimensions
    mutually_exclusive: check_dimensions, no_check_dimensions
no_copy_header: (a boolean)
    Do not copy all of the header from the first file.
    flag: -nocopy_header
    mutually_exclusive: copy_header, no_copy_header
outfiles: (a list of items which are a tuple of the form: (a unicode
    string, a file name))
output_file: (a file name)
    output file
    flag: %s, position: -1
output_illegal: (a boolean)
    Value to write out when an illegal operation is done. Default value:
    1.79769e+308
    flag: -illegal_value
    mutually_exclusive: output_nan, output_zero, output_illegal_value
output_nan: (a boolean)
    Output NaN when an illegal operation is done (default).
    flag: -nan
    mutually_exclusive: output_nan, output_zero, output_illegal_value
output_zero: (a boolean)
    Output zero when an illegal operation is done.
    flag: -zero
    mutually_exclusive: output_nan, output_zero, output_illegal_value
propagate_nan: (a boolean)
    Invalid data in any file at a voxel produces a NaN (default).
    flag: -propagate_nan
quiet: (a boolean)
    Do not print out log messages.
    flag: -quiet
    mutually_exclusive: verbose, quiet
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')

```

```

Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
two: (a boolean)
    Create a MINC 2 output file.
    flag: -2
verbose: (a boolean)
    Print out log messages (default).
    flag: -verbose
    mutually_exclusive: verbose, quiet
voxel_range: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    Valid range for output data.
    flag: -range %d %d

```

Outputs:

```

output_file: (an existing file name)
    output file

```

Convert[Link to code](#)Wraps command **mincconvert**

convert between MINC 1 to MINC 2 format.

Examples

```

>>> from nipy.interfaces.minc import Convert
>>> from nipy.interfaces.minc.testdata import minc2Dfile
>>> c = Convert(input_file=minc2Dfile, output_file='/tmp/out.mnc', two=True) #_
↳ Convert to MINC2 format.
>>> c.run()

```

Inputs:

```

[Mandatory]
input_file: (an existing file name)
    input file for converting
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
chunk: (a long integer >= 0)
    Set the target block size for chunking (0 default, >1 block size).
    flag: -chunk %d
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
compression: (0 or 1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9)
    Set the compression level, from 0 (disabled) to 9 (maximum).
    flag: -compress %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})

```

```

    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_file: (a file name)
    output file
    flag: %s, position: -1
template: (a boolean)
    Create a template file. The dimensions, variables, and attributes of
    the input file are preserved but all data is set to zero.
    flag: -template
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
two: (a boolean)
    Create a MINC 2 output file.
    flag: -2

```

Outputs:

```

output_file: (an existing file name)
    output file

```

Copy

[Link to code](#)**Wraps command `minccopy`**

Copy image values from one MINC file to another. Both the input and output files must exist, and the images in both files must have an equal number of dimensions and equal dimension lengths.

NOTE: This program is intended primarily for use with scripts such as `mincedit`. It does not follow the typical design rules of most MINC command-line tools and therefore should be used only with caution.

Inputs:

```

[Mandatory]
input_file: (an existing file name)
    input file to copy
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_file: (a file name)
    output file
    flag: %s, position: -1
pixel_values: (a boolean)
    Copy pixel values as is.
    flag: -pixel_values
    mutually_exclusive: pixel_values, real_values

```

```
real_values: (a boolean)
    Copy real pixel intensities (default).
    flag: -real_values
    mutually_exclusive: pixel_values, real_values
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
output_file: (an existing file name)
    output file
```

Dump

[Link to code](#)

Wraps command **mincdump**

Dump a MINC file. Typically used in conjunction with mincgen (see Gen).

Examples

```
>>> from nipy.interfaces.minc import Dump
>>> from nipy.interfaces.minc.testdata import minc2Dfile
```

```
>>> dump = Dump(input_file=minc2Dfile)
>>> dump.run()
```

```
>>> dump = Dump(input_file=minc2Dfile, output_file='/tmp/out.txt', precision=(3, 4))
>>> dump.run()
```

Inputs:

```
[Mandatory]
input_file: (an existing file name)
    input file
    flag: %s, position: -2

[Optional]
annotations_brief: ('c' or 'f')
    Brief annotations for C or Fortran indices in data.
    flag: -b %s
    mutually_exclusive: annotations_brief, annotations_full
annotations_full: ('c' or 'f')
    Full annotations for C or Fortran indices in data.
    flag: -f %s
    mutually_exclusive: annotations_brief, annotations_full
args: (a unicode string)
    Additional parameters to the command
    flag: %s
coordinate_data: (a boolean)
    Coordinate variable data and header information.
    flag: -c
    mutually_exclusive: coordinate_data, header_data
environ: (a dictionary with keys which are a bytes or None or a value
```



```

    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
Environment variables
header_data: (a boolean)
    Header information only, no data.
    flag: -h
    mutually_exclusive: coordinate_data, header_data
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
line_length: (a long integer >= 0)
    Line length maximum in data section (default 80).
    flag: -l %d
netcdf_name: (a unicode string)
    Name for netCDF (default derived from file name).
    flag: -n %s
out_file: (a file name)
    flag: > %s, position: -1
output_file: (a file name)
    output file
precision: (an integer (int or long) or a tuple of the form: (an
    integer (int or long), an integer (int or long)))
    Display floating-point values with less precision
    flag: %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
variables: (a list of items which are a unicode string)
    Output data for specified variables only.
    flag: -v %s

```

Outputs:

```

output_file: (an existing file name)
    output file

```

Extract

[Link to code](#)

Wraps command **mincextract**

Dump a hyperslab of MINC file data.

Examples

```

>>> from nipyte.interfaces.minc import Extract
>>> from nipyte.interfaces.minc.testdata import minc2Dfile

```

```

>>> extract = Extract(input_file=minc2Dfile)
>>> extract.run()

```

```

>>> extract = Extract(input_file=minc2Dfile, start=[3, 10, 5], count=[4, 4, 4]) #_
↪extract a 4x4x4 slab at offset [3, 10, 5]
>>> extract.run()

```

Inputs:

```
[Mandatory]
input_file: (an existing file name)
    input file
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
count: (a list of items which are an integer (int or long))
    Specifies edge lengths of hyperslab to read.
    flag: -count %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
flip_any_direction: (a boolean)
    Do not flip images (Default).
    flag: -any_direction
    mutually_exclusive: flip_positive_direction,
        flip_negative_direction, flip_any_direction
flip_negative_direction: (a boolean)
    Flip images to always have negative direction.
    flag: -negative_direction
    mutually_exclusive: flip_positive_direction,
        flip_negative_direction, flip_any_direction
flip_positive_direction: (a boolean)
    Flip images to always have positive direction.
    flag: -positive_direction
    mutually_exclusive: flip_positive_direction,
        flip_negative_direction, flip_any_direction
flip_x_any: (a boolean)
    Don't flip images along x-axis (default).
    flag: -xanydirection
    mutually_exclusive: flip_x_positive, flip_x_negative, flip_x_any
flip_x_negative: (a boolean)
    Flip images to give negative xspace:step value (right-to-left).
    flag: -xdirection
    mutually_exclusive: flip_x_positive, flip_x_negative, flip_x_any
flip_x_positive: (a boolean)
    Flip images to give positive xspace:step value (left-to-right).
    flag: +xdirection
    mutually_exclusive: flip_x_positive, flip_x_negative, flip_x_any
flip_y_any: (a boolean)
    Don't flip images along y-axis (default).
    flag: -yanydirection
    mutually_exclusive: flip_y_positive, flip_y_negative, flip_y_any
flip_y_negative: (a boolean)
    Flip images to give negative yspace:step value (ant-to-post).
    flag: -ydirection
    mutually_exclusive: flip_y_positive, flip_y_negative, flip_y_any
flip_y_positive: (a boolean)
    Flip images to give positive yspace:step value (post-to-ant).
    flag: +ydirection
    mutually_exclusive: flip_y_positive, flip_y_negative, flip_y_any
flip_z_any: (a boolean)
    Don't flip images along z-axis (default).
    flag: -zanydirection
```

```

    mutually_exclusive: flip_z_positive, flip_z_negative, flip_z_any
flip_z_negative: (a boolean)
    Flip images to give negative zspace:step value (sup-to-inf).
    flag: -zdirection
    mutually_exclusive: flip_z_positive, flip_z_negative, flip_z_any
flip_z_positive: (a boolean)
    Flip images to give positive zspace:step value (inf-to-sup).
    flag: +zdirection
    mutually_exclusive: flip_z_positive, flip_z_negative, flip_z_any
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
image_maximum: (a float)
    Specify the maximum real image value for normalization.Default
    value: 1.79769e+308.
    flag: -image_maximum %s
image_minimum: (a float)
    Specify the minimum real image value for normalization.Default
    value: 1.79769e+308.
    flag: -image_minimum %s
image_range: (a tuple of the form: (a float, a float))
    Specify the range of real image values for normalization.
    flag: -image_range %s %s
nonnormalize: (a boolean)
    Turn off pixel normalization.
    flag: -nonnormalize
    mutually_exclusive: normalize, nonnormalize
normalize: (a boolean)
    Normalize integer pixel values to file max and min.
    flag: -normalize
    mutually_exclusive: normalize, nonnormalize
out_file: (a file name)
    flag: > %s, position: -1
output_file: (a file name)
    output file
start: (a list of items which are an integer (int or long))
    Specifies corner of hyperslab (C conventions for indices).
    flag: -start %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
write_ascii: (a boolean)
    Write out data as ascii strings (default).
    flag: -ascii
    mutually_exclusive: write_ascii, write_ascii, write_byte,
    write_short, write_int, write_long, write_float, write_double,
    write_signed, write_unsigned
write_byte: (a boolean)
    Write out data as bytes.
    flag: -byte
    mutually_exclusive: write_ascii, write_ascii, write_byte,
    write_short, write_int, write_long, write_float, write_double,
    write_signed, write_unsigned
write_double: (a boolean)
    Write out data as double precision floating-point values.
    flag: -double
    mutually_exclusive: write_ascii, write_ascii, write_byte,

```

```

        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
write_float: (a boolean)
    Write out data as single precision floating-point values.
    flag: -float
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
write_int: (a boolean)
    Write out data as 32-bit integers.
    flag: -int
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
write_long: (a boolean)
    Superseded by write_int.
    flag: -long
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
write_range: (a tuple of the form: (a float, a float))
    Specify the range of output values
    Default value: 1.79769e+308 1.79769e+308.
    flag: -range %s %s
write_short: (a boolean)
    Write out data as short integers.
    flag: -short
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
write_signed: (a boolean)
    Write out signed data.
    flag: -signed
    mutually_exclusive: write_signed, write_unsigned
write_unsigned: (a boolean)
    Write out unsigned data.
    flag: -unsigned
    mutually_exclusive: write_signed, write_unsigned

```

Outputs:

```

output_file: (an existing file name)
    output file in raw/text format

```

Gennlxfm

[Link to code](#)

Wraps command **gennlxfm**

Generate nonlinear xfms. Currently only identity xfms are supported!

This tool is part of minc-widgets:

<https://github.com/BIC-MNI/minc-widgets/blob/master/gennlxfm/gennlxfm>

Examples

```

>>> from nipy.interfaces.minc import Gennlxfm
>>> from nipy.interfaces.minc.testdata import minc2Dfile

```

```
>>> gennlxfm = Gennlxfm(step=1, like=minc2Dfile)
>>> gennlxfm.run()
```

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ident: (a boolean)
    Generate an identity xfm. Default: False.
    flag: -ident
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
like: (an existing file name)
    Generate a nlxfm like this file.
    flag: -like %s
output_file: (a file name)
    output file
    flag: %s, position: -1
step: (an integer (int or long))
    Output ident xfm step [default: 1].
    flag: -step %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Print out log messages. Default: False.
    flag: -verbose
```

Outputs:

```
output_file: (an existing file name)
    output file
output_grid: (an existing file name)
    output grid
```

Math

[Link to code](#)

Wraps command **mincmath**

Various mathematical operations supplied by mincmath.

Examples

```
>>> from nipy.interfaces.minc import Math
>>> from nipy.interfaces.minc.testdata import minc2Dfile
```

Scale: volume*3.0 + 2:

```
>>> scale = Math(input_files=[minc2Dfile], scale=(3.0, 2))
>>> scale.run()
```

Test if >= 1.5:

```
>>> gt = Math(input_files=[minc2Dfile], test_gt=1.5)
>>> gt.run()
```

Inputs:

```
[Mandatory]
filelist: (a file name)
    Specify the name of a file containing input file names.
    flag: -filelist %s
    mutually_exclusive: input_files, filelist
input_files: (a list of items which are a file name)
    input file(s) for calculation
    flag: %s, position: -2
    mutually_exclusive: input_files, filelist

[Optional]
abs: (a boolean)
    Take absolute value of a volume.
    flag: -abs
args: (a unicode string)
    Additional parameters to the command
    flag: %s
calc_add: (a boolean or a float)
    Add N volumes or volume + constant.
    flag: -add
calc_and: (a boolean)
    Calculate vol1 && vol2 (&& ...).
    flag: -and
calc_div: (a boolean or a float)
    Divide 2 volumes or volume / constant.
    flag: -div
calc_mul: (a boolean or a float)
    Multiply N volumes or volume * constant.
    flag: -mult
calc_not: (a boolean)
    Calculate !vol1.
    flag: -not
calc_or: (a boolean)
    Calculate vol1 || vol2 (|| ...).
    flag: -or
calc_sub: (a boolean or a float)
    Subtract 2 volumes or volume - constant.
    flag: -sub
check_dimensions: (a boolean)
    Check that dimension info matches across files (default).
    flag: -check_dimensions
    mutually_exclusive: check_dimensions, no_check_dimensions
clamp: (a tuple of the form: (a float, a float))
```

```

    Clamp a volume to lie between two values.
    flag: -clamp -const2 %s %s
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
copy_header: (a boolean)
    Copy all of the header from the first file (default for one file).
    flag: -copy_header
    mutually_exclusive: copy_header, no_copy_header
count_valid: (a boolean)
    Count the number of valid values in N volumes.
    flag: -count_valid
dimension: (a unicode string)
    Specify a dimension along which we wish to perform a calculation.
    flag: -dimension %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
exp: (a tuple of the form: (a float, a float))
    Calculate c2*exp(c1*x). Both constants must be specified.
    flag: -exp -const2 %s %s
format_byte: (a boolean)
    Write out byte data.
    flag: -byte
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_double: (a boolean)
    Write out double-precision floating-point data.
    flag: -double
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_filetype: (a boolean)
    Use data type of first file (default).
    flag: -filetype
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_float: (a boolean)
    Write out single-precision floating-point data.
    flag: -float
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_int: (a boolean)
    Write out 32-bit integer data.
    flag: -int
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_long: (a boolean)
    Superseded by -int.
    flag: -long
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned

```

```
format_short: (a boolean)
    Write out short integer data.
    flag: -short
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_signed: (a boolean)
    Write signed integer data.
    flag: -signed
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_unsigned: (a boolean)
    Write unsigned integer data (default).
    flag: -unsigned
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
ignore_nan: (a boolean)
    Ignore invalid data (NaN) for accumulations.
    flag: -ignore_nan
invert: (a float)
    Calculate 1/c.
    flag: -invert -const %s
isnan: (a boolean)
    Test for NaN values in voll.
    flag: -isnan
log: (a tuple of the form: (a float, a float))
    Calculate log(x/c2)/c1. The constants c1 and c2 default to 1.
    flag: -log -const2 %s %s
max_buffer_size_in_kb: (a long integer >= 0)
    Specify the maximum size of the internal buffers (in kbytes).
    flag: -max_buffer_size_in_kb %d
maximum: (a boolean)
    Find maximum of N volumes.
    flag: -maximum
minimum: (a boolean)
    Find minimum of N volumes.
    flag: -minimum
nisnan: (a boolean)
    Negation of -isnan.
    flag: -nisnan
no_check_dimensions: (a boolean)
    Do not check dimension info.
    flag: -nocheck_dimensions
    mutually_exclusive: check_dimensions, no_check_dimensions
no_copy_header: (a boolean)
    Do not copy all of the header from the first file (default for many
    files)).
    flag: -nocopy_header
    mutually_exclusive: copy_header, no_copy_header
nsegment: (a tuple of the form: (a float, a float))
    Opposite of -segment: within range = 0, outside range = 1.
    flag: -nsegment -const2 %s %s
output_file: (a file name)
```



```

    output file
    flag: %s, position: -1
output_illegal: (a boolean)
    Value to write out when an illegal operation is done. Default value:
    1.79769e+308
    flag: -illegal_value
    mutually_exclusive: output_nan, output_zero, output_illegal_value
output_nan: (a boolean)
    Output NaN when an illegal operation is done (default).
    flag: -nan
    mutually_exclusive: output_nan, output_zero, output_illegal_value
output_zero: (a boolean)
    Output zero when an illegal operation is done.
    flag: -zero
    mutually_exclusive: output_nan, output_zero, output_illegal_value
percentdiff: (a float)
    Percent difference between 2 volumes, thresholded (const def=0.0).
    flag: -percentdiff
propagate_nan: (a boolean)
    Invalid data in any file at a voxel produces a NaN (default).
    flag: -propagate_nan
scale: (a tuple of the form: (a float, a float))
    Scale a volume: volume * c1 + c2.
    flag: -scale -const2 %s %s
segment: (a tuple of the form: (a float, a float))
    Segment a volume using range of -const2: within range = 1, outside
    range = 0.
    flag: -segment -const2 %s %s
sqrt: (a boolean)
    Take square root of a volume.
    flag: -sqrt
square: (a boolean)
    Take square of a volume.
    flag: -square
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
test_eq: (a boolean or a float)
    Test for integer vol1 == vol2 or vol1 == constant.
    flag: -eq
test_ge: (a boolean or a float)
    Test for vol1 >= vol2 or vol1 >= const.
    flag: -ge
test_gt: (a boolean or a float)
    Test for vol1 > vol2 or vol1 > constant.
    flag: -gt
test_le: (a boolean or a float)
    Test for vol1 <= vol2 or vol1 <= const.
    flag: -le
test_lt: (a boolean or a float)
    Test for vol1 < vol2 or vol1 < constant.
    flag: -lt
test_ne: (a boolean or a float)
    Test for integer vol1 != vol2 or vol1 != const.
    flag: -ne
two: (a boolean)
    Create a MINC 2 output file.

```

```

    flag: -2
voxel_range: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    Valid range for output data.
    flag: -range %d %d

```

Outputs:

```

output_file: (an existing file name)
    output file

```

NlpFit

[Link to code](#)

Wraps command **nlpfit**

Hierarchical non-linear fitting with blurring.

This tool is part of the minc-widgets package:

<https://github.com/BIC-MNI/minc-widgets/blob/master/nlpfit/nlpfit>

Examples

```

>>> from nipy.interfaces.minc import NlpFit
>>> from nipy.interfaces.minc.testdata import nonempty_minc_data, nlp_config
>>> from nipy.testing import example_data

```

```

>>> source = nonempty_minc_data(0)
>>> target = nonempty_minc_data(1)
>>> source_mask = nonempty_minc_data(2)
>>> config = nlp_config
>>> initial = example_data('minc_initial.xfm')
>>> nlpfit = NlpFit(config_file=config, init_xfm=initial, source_mask=source_mask,
↪ source=source, target=target)
>>> nlpfit.run()

```

Inputs:

```

[Mandatory]
config_file: (an existing file name)
    File containing the fitting configuration use.
    flag: -config_file %s
init_xfm: (an existing file name)
    Initial transformation (default identity).
    flag: -init_xfm %s
source: (an existing file name)
    source Minc file
    flag: %s, position: -3
source_mask: (an existing file name)
    Source mask to use during fitting.
    flag: -source_mask %s
target: (an existing file name)
    target Minc file
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s

```

```

clobber: (a boolean, nipyte default value: True)
    Overwrite existing file.
    flag: -clobber
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
input_grid_files: (a list of items which are a file name)
    input grid file(s)
output_xfm: (a file name)
    output xfm file
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Print out log messages. Default: False.
    flag: -verbose

```

Outputs:

```

output_grid: (an existing file name)
    output grid file
output_xfm: (an existing file name)
    output xfm file

```

Norm[Link to code](#)Wraps command **mincnorm****Normalise a file between a max and minimum (possibly)** using two histogram pct's.**Examples**

```

>>> from nipyte.interfaces.minc import Norm
>>> from nipyte.interfaces.minc.testdata import minc2Dfile
>>> n = Norm(input_file=minc2Dfile, output_file='/tmp/out.mnc') # Normalise the_
↪file.
>>> n.run()

```

Inputs:

```

[Mandatory]
input_file: (an existing file name)
    input file to normalise
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clamp: (a boolean, nipyte default value: True)
    Force the ouput range between limits [default].

```

```
    flag: -clamp
clobber: (a boolean, nipyre default value: True)
    Overwrite existing file.
    flag: -clobber
cutoff: (0.0 <= a floating point number <= 100.0)
    Cutoff value to use to calculate thresholds by a histogram PcT in %.
    [default: 0.01]
    flag: -cutoff %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
lower: (a float)
    Lower real value to use.
    flag: -lower %s
mask: (a file name)
    Calculate the image normalisation within a mask.
    flag: -mask %s
out_ceil: (a float)
    Output files minimum [default: 100]
    flag: -out_ceil %s
out_floor: (a float)
    Output files maximum [default: 0]
    flag: -out_floor %s
output_file: (a file name)
    output file
    flag: %s, position: -1
output_threshold_mask: (a file name)
    File in which to store the threshold mask.
    flag: -threshold_mask %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (a boolean)
    Threshold the image (set values below threshold_perc to -out_floor).
    flag: -threshold
threshold_blur: (a float)
    Blur FWHM for intensity edges then thresholding [default: 2].
    flag: -threshold_blur %s
threshold_bmt: (a boolean)
    Use the resulting image BiModalT as the threshold.
    flag: -threshold_bmt
threshold_perc: (0.0 <= a floating point number <= 100.0)
    Threshold percentage (0.1 == lower 10% of intensity range) [default:
    0.1].
    flag: -threshold_perc %s
upper: (a float)
    Upper real value to use.
    flag: -upper %s
```

Outputs:

```
output_file: (an existing file name)
    output file
```

```
output_threshold_mask: (a file name)
    threshold mask file
```

Pik

[Link to code](#)

Wraps command **mincpik**

Generate images from minc files.

Mincpik uses ImageMagick to generate images from Minc files.

Examples

```
>>> from nipy.interfaces.minc import Pik
>>> from nipy.interfaces.minc.testdata import nonempty_minc_data
```

```
>>> file0 = nonempty_minc_data(0)
>>> pik = Pik(input_file=file0, title='foo')
>>> pik .run()
```

Inputs:

```
[Mandatory]
input_file: (an existing file name)
    input file
    flag: %s, position: -2

[Optional]
annotated_bar: (a boolean)
    create an annotated bar to match the image (use height of the output
    image)
    flag: --anot_bar
args: (a unicode string)
    Additional parameters to the command
    flag: %s
auto_range: (a boolean)
    Automatically determine image range using a 5 and 95% PcT.
    (histogram)
    flag: --auto_range
    mutually_exclusive: image_range, auto_range
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
depth: (8 or 16)
    Bitdepth for resulting image 8 or 16 (MSB machines only!)
    flag: --depth %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
horizontal_triplanar_view: (a boolean)
    Create a horizontal triplanar view.
    flag: --horizontal
    mutually_exclusive: vertical_triplanar_view,
    horizontal_triplanar_view
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
```

```

        interface fails to run
image_range: (a tuple of the form: (a float, a float))
    Range of image values to use for pixel intensity.
    flag: --image_range %s %s
    mutually_exclusive: image_range, auto_range
jpg: (a boolean)
    Output a jpg file.
    mutually_exclusive: jpg, png
lookup: (a unicode string)
    Arguments to pass to minclookup
    flag: --lookup %s
minc_range: (a tuple of the form: (a float, a float))
    Valid range of values for MINC file.
    flag: --range %s %s
output_file: (a file name)
    output file
    flag: %s, position: -1
png: (a boolean)
    Output a png file (default).
    mutually_exclusive: jpg, png
sagittal_offset: (an integer (int or long))
    Offset the sagittal slice from the centre.
    flag: --sagittal_offset %s
sagittal_offset_perc: (0 <= a long integer <= 100)
    Offset the sagittal slice by a percentage from the centre.
    flag: --sagittal_offset_perc %d
scale: (an integer (int or long))
    Scaling factor for resulting image. By default images are output at
    twice their original resolution.
    flag: --scale %s
slice_x: (a boolean)
    Get a sagittal (x) slice.
    flag: -x
    mutually_exclusive: slice_z, slice_y, slice_x
slice_y: (a boolean)
    Get a coronal (y) slice.
    flag: -y
    mutually_exclusive: slice_z, slice_y, slice_x
slice_z: (a boolean)
    Get an axial/transverse (z) slice.
    flag: -z
    mutually_exclusive: slice_z, slice_y, slice_x
start: (an integer (int or long))
    Slice number to get. (note this is in voxel co-ordinates).
    flag: --slice %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tile_size: (an integer (int or long))
    Pixel size for each image in a triplanar.
    flag: --tilesize %s
title: (a boolean or a unicode string)
    flag: %s
title_size: (an integer (int or long))
    Font point size for the title.
    flag: --title_size %s
    requires: title

```

```

triplanar: (a boolean)
    Create a triplanar view of the input file.
    flag: --triplanar
vertical_triplanar_view: (a boolean)
    Create a vertical triplanar view (Default).
    flag: --vertical
    mutually_exclusive: vertical_triplanar_view,
        horizontal_triplanar_view
width: (an integer (int or long))
    Autoscale the resulting image to have a fixed image width (in
    pixels).
    flag: --width %s

```

Outputs:

```

output_file: (an existing file name)
    output image

```

Resample[Link to code](#)Wraps command **mincresample**

Resample a minc file.'

Examples

```

>>> from nipy.interfaces.minc import Resample
>>> from nipy.interfaces.minc.testdata import minc2Dfile
>>> r = Resample(input_file=minc2Dfile, output_file='/tmp/out.mnc') # Resample_
    ↳the file.
>>> r.run()

```

Inputs:

```

[Mandatory]
input_file: (an existing file name)
    input file for resampling
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
coronal_slices: (a boolean)
    Write out coronal slices
    flag: -coronal
    mutually_exclusive: transverse, sagittal, coronal
dircos: (a tuple of the form: (a float, a float, a float))
    Direction cosines along each dimension (X, Y, Z). Default
    value:1.79769e+308 1.79769e+308 1.79769e+308 1.79769e+308 ...
    1.79769e+308 1.79769e+308 1.79769e+308 1.79769e+308 1.79769e+308.
    flag: -dircos %s %s %s
    mutually_exclusive: nelements, nelements_x_y_or_z
environ: (a dictionary with keys which are a bytes or None or a value

```

```
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {}))
    Environment variables
fill: (a boolean)
    Use a fill value for points outside of input volume.
    flag: -fill
    mutually_exclusive: nofill, fill
fill_value: (a float)
    Specify a fill value for points outside of input volume.Default
    value: 1.79769e+308.
    flag: -fillvalue %s
    requires: fill
format_byte: (a boolean)
    Write out byte data.
    flag: -byte
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
format_double: (a boolean)
    Write out double-precision floating-point data.
    flag: -double
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
format_float: (a boolean)
    Write out single-precision floating-point data.
    flag: -float
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
format_int: (a boolean)
    Write out 32-bit integer data.
    flag: -int
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
format_long: (a boolean)
    Superseded by -int.
    flag: -long
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
format_short: (a boolean)
    Write out short integer data.
    flag: -short
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
format_signed: (a boolean)
    Write signed integer data.
    flag: -signed
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
format_unsigned: (a boolean)
    Write unsigned integer data (default).
    flag: -unsigned
    mutually_exclusive: format_byte, format_short, format_int,
```



```

        format_long, format_float, format_double, format_signed,
        format_unsigned
half_width_sinc_window: (5 or 1 or 2 or 3 or 4 or 6 or 7 or 8 or 9 or
10)
    Set half-width of sinc window (1-10). Default value: 5.
    flag: -width %s
    requires: sinc_interpolation
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
input_grid_files: (a list of items which are a file name)
    input grid file(s)
invert_transformation: (a boolean)
    Invert the transformation before using it.
    flag: -invert_transformation
keep_real_range: (a boolean)
    Keep the real scale of the input volume.
    flag: -keep_real_range
    mutually_exclusive: keep_real_range, nokeep_real_range
like: (a file name)
    Specifies a model file for the resampling.
    flag: -like %s
nearest_neighbour_interpolation: (a boolean)
    Do nearest neighbour interpolation.
    flag: -nearest_neighbour
    mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
    nearest_neighbour_interpolation, sinc_interpolation
nelements: (a tuple of the form: (an integer (int or long), an
integer (int or long), an integer (int or long)))
    Number of elements along each dimension (X, Y, Z).
    flag: -nelements %s %s %s
    mutually_exclusive: nelements, nelements_x_y_or_z
no_fill: (a boolean)
    Use value zero for points outside of input volume.
    flag: -nofill
    mutually_exclusive: nofill, fill
no_input_sampling: (a boolean)
    Use the input sampling without transforming (old behaviour).
    flag: -use_input_sampling
    mutually_exclusive: vio_transform, no_input_sampling
nokeep_real_range: (a boolean)
    Do not keep the real scale of the data (default).
    flag: -nokeep_real_range
    mutually_exclusive: keep_real_range, nokeep_real_range
origin: (a tuple of the form: (a float, a float, a float))
    Origin of first pixel in 3D space.Default value: 1.79769e+308
    1.79769e+308 1.79769e+308.
    flag: -origin %s %s %s
output_file: (a file name)
    output file
    flag: %s, position: -1
output_range: (a tuple of the form: (a float, a float))
    Valid range for output data. Default value: -1.79769e+308
    -1.79769e+308.
    flag: -range %s %s
sagittal_slices: (a boolean)
    Write out sagittal slices
    flag: -sagittal

```

```

    mutually_exclusive: transverse, sagittal, coronal
sinc_interpolation: (a boolean)
    Do windowed sinc interpolation.
    flag: -sinc
    mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
        nearest_neighbour_interpolation, sinc_interpolation
sinc_window_hamming: (a boolean)
    Set sinc window type to Hamming.
    flag: -hamming
    mutually_exclusive: sinc_window_hanning, sinc_window_hamming
    requires: sinc_interpolation
sinc_window_hanning: (a boolean)
    Set sinc window type to Hanning.
    flag: -hanning
    mutually_exclusive: sinc_window_hanning, sinc_window_hamming
    requires: sinc_interpolation
spacetype: (a unicode string)
    Set the spacetype attribute to a specified string.
    flag: -spacetype %s
standard_sampling: (a boolean)
    Set the sampling to standard values (step, start and dirs).
    flag: -standard_sampling
start: (a tuple of the form: (a float, a float, a float))
    Start point along each dimension (X, Y, Z). Default value:
    1.79769e+308 1.79769e+308 1.79769e+308.
    flag: -start %s %s %s
    mutually_exclusive: nelements, nelements_x_y_or_z
step: (a tuple of the form: (an integer (int or long), an integer
    (int or long), an integer (int or long)))
    Step size along each dimension (X, Y, Z). Default value: (0, 0, 0).
    flag: -step %s %s %s
    mutually_exclusive: nelements, nelements_x_y_or_z
talairach: (a boolean)
    Output is in Talairach space.
    flag: -talairach
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformation: (a file name)
    File giving world transformation. (Default = identity).
    flag: -transformation %s
transverse_slices: (a boolean)
    Write out transverse slices.
    flag: -transverse
    mutually_exclusive: transverse, sagittal, coronal
tricubic_interpolation: (a boolean)
    Do tricubic interpolation.
    flag: -tricubic
    mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
        nearest_neighbour_interpolation, sinc_interpolation
trilinear_interpolation: (a boolean)
    Do trilinear interpolation.
    flag: -trilinear
    mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
        nearest_neighbour_interpolation, sinc_interpolation
two: (a boolean)
    Create a MINC 2 output file.

```

```

    flag: -2
units: (a unicode string)
    Specify the units of the output sampling.
    flag: -units %s
vio_transform: (a boolean)
    VIO_Transform the input sampling with the transform (default).
    flag: -tfm_input_sampling
    mutually_exclusive: vio_transform, no_input_sampling
xdircos: (a float)
    Direction cosines along the X dimension.Default value: 1.79769e+308
    1.79769e+308 1.79769e+308.
    flag: -xdircos %s
    mutually_exclusive: dircos, dircos_x_y_or_z
    requires: ydircos, zdircos
xnelements: (an integer (int or long))
    Number of elements along the X dimension.
    flag: -xnelements %s
    mutually_exclusive: nelements, nelements_x_y_or_z
    requires: ynelements, znelements
xstart: (a float)
    Start point along the X dimension. Default value: 1.79769e+308.
    flag: -xstart %s
    mutually_exclusive: start, start_x_y_or_z
    requires: ystart, zstart
xstep: (an integer (int or long))
    Step size along the X dimension. Default value: 0.
    flag: -xstep %s
    mutually_exclusive: step, step_x_y_or_z
    requires: ystep, zstep
ydircos: (a float)
    Direction cosines along the Y dimension.Default value: 1.79769e+308
    1.79769e+308 1.79769e+308.
    flag: -ydircos %s
    mutually_exclusive: dircos, dircos_x_y_or_z
    requires: xdircos, zdircos
ynelements: (an integer (int or long))
    Number of elements along the Y dimension.
    flag: -ynelements %s
    mutually_exclusive: nelements, nelements_x_y_or_z
    requires: xnelements, znelements
ystart: (a float)
    Start point along the Y dimension. Default value: 1.79769e+308.
    flag: -ystart %s
    mutually_exclusive: start, start_x_y_or_z
    requires: xstart, zstart
ystep: (an integer (int or long))
    Step size along the Y dimension. Default value: 0.
    flag: -ystep %s
    mutually_exclusive: step, step_x_y_or_z
    requires: xstep, zstep
zdircos: (a float)
    Direction cosines along the Z dimension.Default value: 1.79769e+308
    1.79769e+308 1.79769e+308.
    flag: -zdircos %s
    mutually_exclusive: dircos, dircos_x_y_or_z
    requires: xdircos, ydircos
znelements: (an integer (int or long))
    Number of elements along the Z dimension.

```

```
    flag: -znelements %s
    mutually_exclusive: nelements, nelements_x_y_or_z
    requires: xnelements, ynelements
zstart: (a float)
    Start point along the Z dimension. Default value: 1.79769e+308.
    flag: -zstart %s
    mutually_exclusive: start, start_x_y_or_z
    requires: xstart, ystart
zstep: (an integer (int or long))
    Step size along the Z dimension. Default value: 0.
    flag: -zstep %s
    mutually_exclusive: step, step_x_y_or_z
    requires: xstep, ystep
```

Outputs:

```
output_file: (an existing file name)
    output file
```

Reshape

[Link to code](#)

Wraps command **mincreshape**

Cut a hyperslab out of a minc file, with dimension reordering.

This is also useful for rewriting with a different format, for example converting to short (see example below).

Examples

```
>>> from nipyne.interfaces.minc import Reshape
>>> from nipyne.interfaces.minc.testdata import nonempty_minc_data
```

```
>>> input_file = nonempty_minc_data(0)
>>> reshape_to_short = Reshape(input_file=input_file, write_short=True)
>>> reshape_to_short.run()
```

Inputs:

```
[Mandatory]
input_file: (a file name)
    input file
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clobber: (a boolean, nipyne default value: True)
    Overwrite existing file.
    flag: -clobber
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_file: (a file name)
```

```

    output file
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Print out log messages. Default: False.
    flag: -verbose
write_short: (a boolean)
    Convert to short integer data.
    flag: -short

```

Outputs:

```

output_file: (an existing file name)
    output file

```

ToEcat[Link to code](#)Wraps command **minctoeocat**

Convert a 2D image, a 3D volumes or a 4D dynamic volumes written in MINC file format to a 2D, 3D or 4D Ecat7 file.

Examples

```

>>> from nipy.interfaces.minc import ToEcat
>>> from nipy.interfaces.minc.testdata import minc2Dfile

```

```

>>> c = ToEcat(input_file=minc2Dfile)
>>> c.run()

```

```

>>> c = ToEcat(input_file=minc2Dfile, voxels_as_integers=True)
>>> c.run()

```

Inputs:

```

[Mandatory]
input_file: (an existing file name)
    input file to convert
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_acquisition_variable: (a boolean)
    Ignore informations from the minc acquisition variable.
    flag: -ignore_acquisition_variable
ignore_ecat_acquisition_variable: (a boolean)
    Ignore informations from the minc ecat_acquisition variable.

```

```

    flag: -ignore_ecat_acquisition_variable
ignore_ecat_main: (a boolean)
    Ignore informations from the minc ecat-main variable.
    flag: -ignore_ecat_main
ignore_ecat_subheader_variable: (a boolean)
    Ignore informations from the minc ecat-subhdr variable.
    flag: -ignore_ecat_subheader_variable
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
ignore_patient_variable: (a boolean)
    Ignore informations from the minc patient variable.
    flag: -ignore_patient_variable
ignore_study_variable: (a boolean)
    Ignore informations from the minc study variable.
    flag: -ignore_study_variable
no_decay_corr_fctr: (a boolean)
    Do not compute the decay correction factors
    flag: -no_decay_corr_fctr
output_file: (a file name)
    output file
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
voxels_as_integers: (a boolean)
    Voxel values are treated as integers, scale and calibration factors
    are set to unity
    flag: -label

```

Outputs:

```

output_file: (an existing file name)
    output file

```

ToRaw[Link to code](#)Wraps command **minctoraw**

Dump a chunk of MINC file data. This program is largely superceded by mincextract (see Extract).

Examples

```

>>> from nipy.interfaces.minc import ToRaw
>>> from nipy.interfaces.minc.testdata import minc2Dfile

```

```

>>> toraw = ToRaw(input_file=minc2Dfile)
>>> toraw.run()

```

```

>>> toraw = ToRaw(input_file=minc2Dfile, write_range=(0, 100))
>>> toraw.run()

```

Inputs:

```

[Mandatory]
input_file: (an existing file name)
    input file
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
nonnormalize: (a boolean)
    Turn off pixel normalization.
    flag: -nonnormalize
    mutually_exclusive: normalize, nonnormalize
normalize: (a boolean)
    Normalize integer pixel values to file max and min.
    flag: -normalize
    mutually_exclusive: normalize, nonnormalize
out_file: (a file name)
    flag: > %s, position: -1
output_file: (a file name)
    output file
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
write_byte: (a boolean)
    Write out data as bytes.
    flag: -byte
    mutually_exclusive: write_byte, write_short, write_int, write_long,
    write_float, write_double
write_double: (a boolean)
    Write out data as double precision floating-point values.
    flag: -double
    mutually_exclusive: write_byte, write_short, write_int, write_long,
    write_float, write_double
write_float: (a boolean)
    Write out data as single precision floating-point values.
    flag: -float
    mutually_exclusive: write_byte, write_short, write_int, write_long,
    write_float, write_double
write_int: (a boolean)
    Write out data as 32-bit integers.
    flag: -int
    mutually_exclusive: write_byte, write_short, write_int, write_long,
    write_float, write_double
write_long: (a boolean)
    Superseded by write_int.
    flag: -long
    mutually_exclusive: write_byte, write_short, write_int, write_long,
    write_float, write_double
write_range: (a tuple of the form: (a float, a float))

```

```

        Specify the range of output values.Default value: 1.79769e+308
        1.79769e+308.
        flag: -range %s %s
write_short: (a boolean)
    Write out data as short integers.
    flag: -short
    mutually_exclusive: write_byte, write_short, write_int, write_long,
        write_float, write_double
write_signed: (a boolean)
    Write out signed data.
    flag: -signed
    mutually_exclusive: write_signed, write_unsigned
write_unsigned: (a boolean)
    Write out unsigned data.
    flag: -unsigned
    mutually_exclusive: write_signed, write_unsigned

```

Outputs:

```

output_file: (an existing file name)
    output file in raw format

```

VolSymm

[Link to code](#)Wraps command **volsymm**

Make a volume symmetric about an axis either linearly and/or nonlinearly. This is done by registering a volume to a flipped image of itself.

This tool is part of the minc-widgets package:

<https://github.com/BIC-MNI/minc-widgets/blob/master/volsymm/volsymm>

Examples

```

>>> from nipy.interfaces.minc import VolSymm
>>> from nipy.interfaces.minc.testdata import nonempty_minc_data

```

```

>>> input_file = nonempty_minc_data(0)
>>> volsymm = VolSymm(input_file=input_file)
>>> volsymm.run()

```

Inputs:

```

[Mandatory]
input_file: (a file name)
    input file
    flag: %s, position: -3

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
config_file: (an existing file name)
    File containing the fitting configuration (nlpfit -help for info).
    flag: -config_file %s

```



```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipyre default value: {})
         Environment variables
fit_linear: (a boolean)
            Fit using a linear xfm.
            flag: -linear
fit_nonlinear: (a boolean)
              Fit using a non-linear xfm.
              flag: -nonlinear
ignore_exception: (a boolean, nipyre default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
input_grid_files: (a list of items which are a file name)
                  input grid file(s)
nofit: (a boolean)
       Use the input transformation instead of generating one.
       flag: -nofit
output_file: (a file name)
             output file
             flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
trans_file: (a file name)
            output xfm trans file
            flag: %s, position: -2
verbose: (a boolean)
         Print out log messages. Default: False.
         flag: -verbose
x: (a boolean)
   Flip volume in x-plane (default).
   flag: -x
y: (a boolean)
   Flip volume in y-plane.
   flag: -y
z: (a boolean)
   Flip volume in z-plane.
   flag: -z

```

Outputs:

```

output_file: (an existing file name)
             output file
output_grid: (an existing file name)
             output grid file
trans_file: (an existing file name)
            xfm trans file

```

Volcentre[Link to code](#)Wraps command **volcentre**

Centre a MINC image's sampling about a point, typically (0,0,0).

Example

```
>>> from nipy.interfaces.minc import Volcentre
>>> from nipy.interfaces.minc.testdata import minc2Dfile
>>> vc = Volcentre(input_file=minc2Dfile)
>>> vc.run()
```

Inputs:

```
[Mandatory]
input_file: (an existing file name)
            input file to centre
            flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
centre: (a tuple of the form: (a float, a float, a float))
        Centre to use (x,y,z) [default: 0 0 0].
        flag: -centre %s %s %s
clobber: (a boolean, nipy default value: True)
          Overwrite existing file.
          flag: -clobber
com: (a boolean)
      Use the CoM of the volume for the new centre (via mincstats).
      Default: False
      flag: -com
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
output_file: (a file name)
              output file
              flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
          Print out log messages. Default: False.
          flag: -verbose
zero_diracos: (a boolean)
               Set the direction cosines to identity [default].
               flag: -zero_diracos
```

Outputs:

```
output_file: (an existing file name)
              output file
```

Voliso

[Link to code](#)

Wraps command **voliso**

Changes the steps and starts in order that the output volume has isotropic sampling.

Examples

```
>>> from nipyre.interfaces.minc import Voliso
>>> from nipyre.interfaces.minc.testdata import minc2Dfile
>>> viso = Voliso(input_file=minc2Dfile, minstep=0.1, avgstep=True)
>>> viso.run()
```

Inputs:

```
[Mandatory]
input_file: (an existing file name)
    input file to convert to isotropic sampling
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
avgstep: (a boolean)
    Calculate the maximum step from the average steps of the input
    volume.
    flag: --avgstep
clobber: (a boolean, nipyre default value: True)
    Overwrite existing file.
    flag: --clobber
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
maxstep: (a float)
    The target maximum step desired in the output volume.
    flag: --maxstep %s
minstep: (a float)
    The target minimum step desired in the output volume.
    flag: --minstep %s
output_file: (a file name)
    output file
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Print out log messages. Default: False.
    flag: --verbose
```

Outputs:

```
output_file: (an existing file name)
    output file
```

Volpad

[Link to code](#)

Wraps command **volpad**

Centre a MINC image's sampling about a point, typically (0,0,0).

Examples

```
>>> from nipy.interfaces.minc import Volpad
>>> from nipy.interfaces.minc.testdata import minc2Dfile
>>> vp = Volpad(input_file=minc2Dfile, smooth=True, smooth_distance=4)
>>> vp.run()
```

Inputs:

```
[Mandatory]
input_file: (an existing file name)
    input file to centre
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
auto: (a boolean)
    Automatically determine padding distances (uses -distance as max).
    Default: False.
    flag: -auto
auto_freq: (a float)
    Frequency of voxels over bimodal threshold to stop at [default:
    500].
    flag: -auto_freq %s
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
distance: (an integer (int or long))
    Padding distance (in voxels) [default: 4].
    flag: -distance %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_file: (a file name)
    output file
    flag: %s, position: -1
smooth: (a boolean)
    Smooth (blur) edges before padding. Default: False.
    flag: -smooth
smooth_distance: (an integer (int or long))
    Smoothing distance (in voxels) [default: 4].
    flag: -smooth_distance %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Print out log messages. Default: False.
    flag: -verbose
```

Outputs:

```
output_file: (an existing file name)
             output file
```

XfmAvg

[Link to code](#)

Wraps command **xfmavg**

Average a number of xfm transforms using matrix logs and exponents. The program xfmavg calls Octave for numerical work.

This tool is part of the minc-widgets package:

<https://github.com/BIC-MNI/minc-widgets/tree/master/xfmavg>

Examples

```
>>> from nipy.interfaces.minc import XfmAvg
>>> from nipy.interfaces.minc.testdata import nonempty_minc_data, nlp_config
>>> from nipy.testing import example_data
```

```
>>> xfm1 = example_data('minc_initial.xfm')
>>> xfm2 = example_data('minc_initial.xfm') # cheating for doctest
>>> xfmavg = XfmAvg(input_files=[xfm1, xfm2])
>>> xfmavg.run()
```

Inputs:

```
[Mandatory]
input_files: (a list of items which are a file name)
             input file(s)
             flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
avg_linear: (a boolean)
            average the linear part [default].
            flag: -avg_linear
avg_nonlinear: (a boolean)
              average the non-linear part [default].
              flag: -avg_nonlinear
clobber: (a boolean, nipy default value: True)
         Overwrite existing file.
         flag: -clobber
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
ignore_linear: (a boolean)
               opposite of -avg_linear.
               flag: -ignore_linear
ignore_nonlinear: (a boolean)
                  opposite of -avg_nonlinear.
                  flag: -ignore_nonlinear
```

```
input_grid_files: (a list of items which are a file name)
    input grid file(s)
output_file: (a file name)
    output file
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Print out log messages. Default: False.
    flag: -verbose
```

Outputs:

```
output_file: (an existing file name)
    output file
output_grid: (an existing file name)
    output grid file
```

XfmConcat

[Link to code](#)

Wraps command **xfmconcat**

Concatenate transforms together. The output transformation is equivalent to applying input1.xfm, then input2.xfm, ..., in that order.

Examples

```
>>> from nipy.interfaces.minc import XfmConcat
>>> from nipy.interfaces.minc.testdata import minc2Dfile
>>> conc = XfmConcat(input_files=['input1.xfm', 'input1.xfm'])
>>> conc.run()
```

Inputs:

```
[Mandatory]
input_files: (a list of items which are a file name)
    input file(s)
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
input_grid_files: (a list of items which are a file name)
    input grid file(s)
```

```

output_file: (a file name)
    output file
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Print out log messages. Default: False.
    flag: -verbose

```

Outputs:

```

output_file: (an existing file name)
    output file
output_grids: (a list of items which are an existing file name)
    output grids

```

XfmInvert[Link to code](#)Wraps command **xfminvert**

Invert an xfm transform file.

Examples

```

>>> from nipy.interfaces.minc import XfmAvg
>>> from nipy.testing import example_data

```

```

>>> xfm = example_data('minc_initial.xfm')
>>> invert = XfmInvert(input_file=xfm)
>>> invert.run()

```

Inputs:

```

[Mandatory]
input_file: (a file name)
    input file
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    flag: -clobber
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_file: (a file name)
    output file
    flag: %s, position: -1

```

```
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Print out log messages. Default: False.
flag: -verbose
```

Outputs:

```
output_file: (an existing file name)
    output file
output_grid: (an existing file name)
    output grid file
```

interfaces.minc.testdata

nonempty_minc_data()

[Link to code](#)

interfaces.mipav.developer

JistBrainMgdmSegmentation

[Link to code](#)

Wraps command ******java edu.jhu.ece.iacI.jist.cli.run de.mpg.cbs.jist.brain.JistBrainMgdmSegmentation ******

title: MGDM Whole Brain Segmentation

category: Developer Tools

description: Estimate brain structures from an atlas for a MRI dataset (multiple input combinations are possible).

version: 2.0.RC

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inAdjust: ('true' or 'false')
    Adjust intensity priors
    flag: --inAdjust %s
inAtlas: (an existing file name)
    Atlas file
    flag: --inAtlas %s
inCompute: ('true' or 'false')
    Compute posteriors
    flag: --inCompute %s
inCurvature: (a float)
    Curvature weight
    flag: --inCurvature %f
```

```
inData: (a float)
    Data weight
    flag: --inData %f
inFLAIR: (an existing file name)
    FLAIR Image
    flag: --inFLAIR %s
inMP2RAGE: (an existing file name)
    MP2RAGE T1 Map Image
    flag: --inMP2RAGE %s
inMP2RAGE2: (an existing file name)
    MP2RAGE T1-weighted Image
    flag: --inMP2RAGE2 %s
inMPRAGE: (an existing file name)
    MPRAGE T1-weighted Image
    flag: --inMPRAGE %s
inMax: (an integer (int or long))
    Max iterations
    flag: --inMax %d
inMin: (a float)
    Min change
    flag: --inMin %f
inOutput: ('segmentation' or 'memberships')
    Output images
    flag: --inOutput %s
inPV: (an existing file name)
    PV / Dura Image
    flag: --inPV %s
inPosterior: (a float)
    Posterior scale (mm)
    flag: --inPosterior %f
inSteps: (an integer (int or long))
    Steps
    flag: --inSteps %d
inTopology: ('26/6' or '6/26' or '18/6' or '6/18' or '6/6' or 'wcs'
    or 'wco' or 'no')
    Topology
    flag: --inTopology %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outLevelset: (a boolean or a file name)
    Levelset Boundary Image
    flag: --outLevelset %s
outPosterior2: (a boolean or a file name)
    Posterior Maximum Memberships (4D)
    flag: --outPosterior2 %s
outPosterior3: (a boolean or a file name)
    Posterior Maximum Labels (4D)
    flag: --outPosterior3 %s
outSegmented: (a boolean or a file name)
    Segmented Brain Image
    flag: --outSegmented %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
```

```

    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outLevelset: (an existing file name)
    Levelset Boundary Image
outPosterior2: (an existing file name)
    Posterior Maximum Memberships (4D)
outPosterior3: (an existing file name)
    Posterior Maximum Labels (4D)
outSegmented: (an existing file name)
    Segmented Brain Image

```

JistBrainMp2rageDuraEstimation[Link to code](#)

Wraps command ****java edu.jhu.ece.iac1.jist.cli.run de.mpg.cbs.jist.brain.JistBrainMp2rageDuraEstimation ****

title: MP2RAGE Dura Estimation

category: Developer Tools

description: Filters a MP2RAGE brain image to obtain a probability map of dura matter.

version: 3.0.RC

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inDistance: (a float)
    Distance to background (mm)
    flag: --inDistance %f
inSecond: (an existing file name)
    Second inversion (Inv2) Image
    flag: --inSecond %s
inSkull: (an existing file name)
    Skull Stripping Mask
    flag: --inSkull %s
inoutput: ('dura_region' or 'boundary' or 'dura_prior' or 'bg_prior'
    or 'intens_prior')
    Outputs an estimate of the dura / CSF boundary or an estimate of the
    entire dura region.
    flag: --inoutput %s
null: (a unicode string)

```

```

    Execution Time
    flag: --null %s
outDura: (a boolean or a file name)
    Dura Image
    flag: --outDura %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outDura: (an existing file name)
    Dura Image

```

JistBrainMp2rageSkullStripping[Link to code](#)

Wraps command ******java edu.jhu.ece.iac.jist.cli.run de.mpg.cbs.jist.brain.JistBrainMp2rageSkullStripping ******

title: MP2RAGE Skull Stripping

category: Developer Tools

description: Estimate a brain mask for a MP2RAGE dataset. At least a T1-weighted or a T1 map image is required.

version: 3.0.RC

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inFilter: (an existing file name)
    Filter Image (opt)
    flag: --inFilter %s
inSecond: (an existing file name)
    Second inversion (Inv2) Image
    flag: --inSecond %s
inSkip: ('true' or 'false')
    Skip zero values
    flag: --inSkip %s

```

```

inT1: (an existing file name)
    T1 Map (T1_Images) Image (opt)
    flag: --inT1 %s
inT1weighted: (an existing file name)
    T1-weighted (UNI) Image (opt)
    flag: --inT1weighted %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outBrain: (a boolean or a file name)
    Brain Mask Image
    flag: --outBrain %s
outMasked: (a boolean or a file name)
    Masked T1 Map Image
    flag: --outMasked %s
outMasked2: (a boolean or a file name)
    Masked T1-weighted Image
    flag: --outMasked2 %s
outMasked3: (a boolean or a file name)
    Masked Filter Image
    flag: --outMasked3 %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outBrain: (an existing file name)
    Brain Mask Image
outMasked: (an existing file name)
    Masked T1 Map Image
outMasked2: (an existing file name)
    Masked T1-weighted Image
outMasked3: (an existing file name)
    Masked Filter Image

```

JistBrainPartialVolumeFilter[Link to code](#)Wraps command ******java edu.jhu.ece.iac.jist.cli.run de.mpg.cbs.jist.brain.JistBrainPartialVolumeFilter ******

title: Partial Volume Filter

category: Developer Tools

description: Filters an image for regions of partial voluming assuming a ridge-like model of intensity.

version: 2.0.RC

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inInput: (an existing file name)
    Input Image
    flag: --inInput %s
inPV: ('bright' or 'dark' or 'both')
    Outputs the raw intensity values or a probability score for the
    partial volume regions.
    flag: --inPV %s
inoutput: ('probability' or 'intensity')
    output
    flag: --inoutput %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outPartial: (a boolean or a file name)
    Partial Volume Image
    flag: --outPartial %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outPartial: (an existing file name)
    Partial Volume Image

```

JistCortexSurfaceMeshInflation[Link to code](#)

Wraps command ******java edu.jhu.ece.iac.jist.cli.run de.mpg.cbs.jist.cortex.JistCortexSurfaceMeshInflation ******

title: Surface Mesh Inflation

category: Developer Tools

description: Inflates a cortical surface mesh. D. Tosun, M. E. Rettmann, X. Han, X. Tao, C. Xu, S. M. Resnick, D. Pham, and J. L. Prince, Cortical Surface Segmentation and Mapping, NeuroImage, vol. 23, pp. S108–S118, 2004.

version: 3.0.RC

contributor: Duygu Tosun

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inLevelset: (an existing file name)
    Levelset Image
    flag: --inLevelset %s
inLorentzian: ('true' or 'false')
    Lorentzian Norm
    flag: --inLorentzian %s
inMax: (an integer (int or long))
    Max Iterations
    flag: --inMax %d
inMean: (a float)
    Mean Curvature Threshold
    flag: --inMean %f
inSOR: (a float)
    SOR Parameter
    flag: --inSOR %f
inStep: (an integer (int or long))
    Step Size
    flag: --inStep %d
inTopology: ('26/6' or '6/26' or '18/6' or '6/18' or '6/6' or 'wcs'
    or 'wco' or 'no')
    Topology
    flag: --inTopology %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outInflated: (a boolean or a file name)
    Inflated Surface
    flag: --outInflated %s
outOriginal: (a boolean or a file name)
    Original Surface
    flag: --outOriginal %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipyre default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
```

```
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s
```

Outputs:

```
outInflated: (an existing file name)
    Inflated Surface
outOriginal: (an existing file name)
    Original Surface
```

JistIntensityMp2rageMasking[Link to code](#)

Wraps command ******java edu.jhu.ece.iacI.jist.cli.run de.mpg.cbs.jist.intensity.JistIntensityMp2rageMasking ******

title: MP2RAGE Background Masking

category: Developer Tools

description: Estimate a background signal mask for a MP2RAGE dataset.

version: 3.0.RC

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inBackground: ('exponential' or 'half-normal')
    Model distribution for background noise (default is half-normal,
    exponential is more stringent).
    flag: --inBackground %s
inMasking: ('binary' or 'proba')
    Whether to use a binary threshold or a weighted average based on the
    probability.
    flag: --inMasking %s
inQuantitative: (an existing file name)
    Quantitative T1 Map (T1_Images) Image
    flag: --inQuantitative %s
inSecond: (an existing file name)
    Second inversion (Inv2) Image
    flag: --inSecond %s
inSkip: ('true' or 'false')
    Skip zero values
    flag: --inSkip %s
inT1weighted: (an existing file name)
    T1-weighted (UNI) Image
    flag: --inT1weighted %s
null: (a unicode string)
    Execution Time
    flag: --null %s
```



```

outMasked: (a boolean or a file name)
    Masked T1 Map Image
    flag: --outMasked %s
outMasked2: (a boolean or a file name)
    Masked Iso Image
    flag: --outMasked2 %s
outSignal: (a boolean or a file name)
    Signal Proba Image
    flag: --outSignal %s
outSignal2: (a boolean or a file name)
    Signal Mask Image
    flag: --outSignal2 %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outMasked: (an existing file name)
    Masked T1 Map Image
outMasked2: (an existing file name)
    Masked Iso Image
outSignal: (an existing file name)
    Signal Proba Image
outSignal2: (an existing file name)
    Signal Mask Image

```

JistLaminarProfileCalculator[Link to code](#)

Wraps command ******java edu.jhu.ece.iac.jist.cli.run de.mpg.cbs.jist.laminar.JistLaminarProfileCalculator ******

title: Profile Calculator

category: Developer Tools

description: Compute various moments for intensities mapped along a cortical profile.

version: 3.0.RC

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inIntensity: (an existing file name)
    Intensity Profile Image
    flag: --inIntensity %s
inMask: (an existing file name)
    Mask Image (opt, 3D or 4D)
    flag: --inMask %s
incomputed: ('mean' or 'stdev' or 'skewness' or 'kurtosis')
    computed statistic
    flag: --incomputed %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outResult: (a boolean or a file name)
    Result
    flag: --outResult %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outResult: (an existing file name)
    Result

```

JistLaminarProfileGeometry[Link to code](#)Wraps command ******java edu.jhu.ece.iac.jist.cli.run de.mpg.cbs.jist.laminar.JistLaminarProfileGeometry ******

title: Profile Geometry

category: Developer Tools

description: Compute various geometric quantities for a cortical layers.

version: 3.0.RC

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inProfile: (an existing file name)
    Profile Surface Image
    flag: --inProfile %s
incomputed: ('thickness' or 'curvedness' or 'shape_index' or
    'mean_curvature' or 'gauss_curvature' or 'profile_length' or
    'profile_curvature' or 'profile_torsion')
    computed measure
    flag: --incomputed %s
inoutside: (a float)
    outside extension (mm)
    flag: --inoutside %f
inregularization: ('none' or 'Gaussian')
    regularization
    flag: --inregularization %s
insmoothing: (a float)
    smoothing parameter
    flag: --insmoothing %f
null: (a unicode string)
    Execution Time
    flag: --null %s
outResult: (a boolean or a file name)
    Result
    flag: --outResult %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outResult: (an existing file name)
    Result

```

JistLaminarProfileSampling[Link to code](#)

Wraps command ******java edu.jhu.ece.iac.jist.cli.run de.mpg.cbs.jist.laminar.JistLaminarProfileSampling ******

title: Profile Sampling

category: Developer Tools

description: Sample some intensity image along a cortical profile across layer surfaces.

version: 3.0.RC

Inputs:

```
[Mandatory]
```

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inCortex: (an existing file name)
    Cortex Mask (opt)
    flag: --inCortex %s
inIntensity: (an existing file name)
    Intensity Image
    flag: --inIntensity %s
inProfile: (an existing file name)
    Profile Surface Image
    flag: --inProfile %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outProfile2: (a boolean or a file name)
    Profile 4D Mask
    flag: --outProfile2 %s
outProfilemapped: (a boolean or a file name)
    Profile-mapped Intensity Image
    flag: --outProfilemapped %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outProfile2: (an existing file name)
    Profile 4D Mask
outProfilemapped: (an existing file name)
    Profile-mapped Intensity Image

```

JistLaminarROIAveraging[Link to code](#)Wraps command ******java edu.jhu.ece.iacl.jist.cli.run de.mpg.cbs.jist.laminar.JistLaminarROIAveraging ******

title: Profile ROI Averaging

category: Developer Tools

description: Compute an average profile over a given ROI.

version: 3.0.RC

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inIntensity: (an existing file name)
    Intensity Profile Image
    flag: --inIntensity %s
inMask: (an existing file name)
    Mask Image (opt, 3D or 4D)
    flag: --inMask %s
inROI: (an existing file name)
    ROI Mask
    flag: --inROI %s
inROI2: (a unicode string)
    ROI Name
    flag: --inROI2 %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outROI3: (a boolean or a file name)
    ROI Average
    flag: --outROI3 %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outROI3: (an existing file name)
    ROI Average

```

JistLaminarVolumetricLayering[Link to code](#)Wraps command ******java edu.jhu.ece.iac.jist.cli.run de.mpg.cbs.jist.laminar.JistLaminarVolumetricLayering ******

title: Volumetric Layering

category: Developer Tools

description: Builds a continuous layering of the cortex following distance-preserving or volume-preserving models of cortical folding. Wahnert MD, Dinse J, Weiss M, Streicher MN, Wahnert P, Geyer S, Turner R, Bazin PL, Anatomically motivated modeling of cortical laminae, Neuroimage, 2013.

version: 3.0.RC

contributor: Miriam Wahnert (wahnert@cbs.mpg.de) <http://www.cbs.mpg.de/>

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inInner: (an existing file name)
    Inner Distance Image (GM/WM boundary)
    flag: --inInner %s
inLayering: ('distance-preserving' or 'volume-preserving')
    Layering method
    flag: --inLayering %s
inLayering2: ('outward' or 'inward')
    Layering direction
    flag: --inLayering2 %s
inMax: (an integer (int or long))
    Max iterations for narrow band evolution
    flag: --inMax %d
inMin: (a float)
    Min change ratio for narrow band evolution
    flag: --inMin %f
inNumber: (an integer (int or long))
    Number of layers
    flag: --inNumber %d
inOuter: (an existing file name)
    Outer Distance Image (CSF/GM boundary)
    flag: --inOuter %s
inTopology: ('26/6' or '6/26' or '18/6' or '6/18' or '6/6' or 'wcs'
    or 'wco' or 'no')
    Topology
    flag: --inTopology %s
incurvature: (an integer (int or long))
    curvature approximation scale (voxels)
    flag: --incurvature %d
inpresmooth: ('true' or 'false')
    pre-smooth cortical surfaces
    flag: --inpresmooth %s
inratio: (a float)
    ratio smoothing kernel size (voxels)
    flag: --inratio %f
null: (a unicode string)
    Execution Time
    flag: --null %s
outContinuous: (a boolean or a file name)
```

```

Continuous depth measurement
flag: --outContinuous %s
outDiscrete: (a boolean or a file name)
Discrete sampled layers
flag: --outDiscrete %s
outLayer: (a boolean or a file name)
Layer boundary surfaces
flag: --outLayer %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
Set default maximum heap size
flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
Set default maximum number of processes.
flag: -xMaxProcess %d
xPrefExt: ('nrrd')
Output File Type
flag: --xPrefExt %s

```

Outputs:

```

outContinuous: (an existing file name)
Continuous depth measurement
outDiscrete: (an existing file name)
Discrete sampled layers
outLayer: (an existing file name)
Layer boundary surfaces

```

MedicAlgorithmImageCalculator[Link to code](#)

Wraps command ******java edu.jhu.ece.iacI.jist.cli.run edu.jhu.ece.iacI.plugins.utilities.math.MedicAlgorithmImageCalculator ******

title: Image Calculator

category: Developer Tools

description: Perform simple image calculator operations on two images. The operations include ‘Add’, ‘Subtract’, ‘Multiply’, and ‘Divide’

version: 1.10.RC

documentation-url: <http://www.iacI.ece.jhu.edu/>

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
Additional parameters to the command
flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a value
of class 'str', nipy default value: {})
Environment variables
ignore_exception: (a boolean, nipy default value: False)
Print an error message instead of throwing an exception in case the
interface fails to run

```

```

inOperation: ('Add' or 'Subtract' or 'Multiply' or 'Divide' or 'Min'
             or 'Max')
             Operation
             flag: --inOperation %s
inVolume: (an existing file name)
           Volume 1
           flag: --inVolume %s
inVolume2: (an existing file name)
           Volume 2
           flag: --inVolume2 %s
null: (a unicode string)
      Execution Time
      flag: --null %s
outResult: (a boolean or a file name)
           Result Volume
           flag: --outResult %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
             Set default maximum heap size
             flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
             Set default maximum number of processes.
             flag: -xMaxProcess %d
xPrefExt: ('nrrd')
           Output File Type
           flag: --xPrefExt %s

```

Outputs:

```

outResult: (an existing file name)
           Result Volume

```

MedicAlgorithmLesionToads[Link to code](#)

Wraps command ******java edu.jhu.ece.iacI.jist.cli.run edu.jhu.ece.iacI.plugins.classification.MedicAlgorithmLesionToads ******

title: Lesion TOADS

category: Developer Tools

description: Algorithm for simultaneous brain structures and MS lesion segmentation of MS Brains. The brain segmentation is topologically consistent and the algorithm can use multiple MR sequences as input data. N. Shiee, P.-L. Bazin, A.Z. Ozturk, P.A. Calabresi, D.S. Reich, D.L. Pham, "A Topology-Preserving Approach to the Segmentation of Brain Images with Multiple Sclerosis", NeuroImage, vol. 49, no. 2, pp. 1524-1535, 2010.

version: 1.9.R

contributor: Navid Shiee (navid.shiee@nih.gov) <http://iacI.ece.jhu.edu/~nshiee/>

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value

```



```

        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inAtlas: ('With Lesion' or 'No Lesion')
    Atlas to Use
    flag: --inAtlas %s
inAtlas2: (an existing file name)
    Atlas File - With Lesions
    flag: --inAtlas2 %s
inAtlas3: (an existing file name)
    Atlas File - No Lesion - T1 and FLAIR
    flag: --inAtlas3 %s
inAtlas4: (an existing file name)
    Atlas File - No Lesion - T1 Only
    flag: --inAtlas4 %s
inAtlas5: (a float)
    Controls the effect of the statistical atlas on the segmentation
    flag: --inAtlas5 %f
inAtlas6: ('rigid' or 'multi_fully_affine')
    Atlas alignment
    flag: --inAtlas6 %s
inConnectivity: ('(26,6)' or '(6,26)' or '(6,18)' or '(18,6)')
    Connectivity (foreground,background)
    flag: --inConnectivity %s
inCorrect: ('true' or 'false')
    Correct MR field inhomogeneity.
    flag: --inCorrect %s
inFLAIR: (an existing file name)
    FLAIR Image
    flag: --inFLAIR %s
inInclude: ('true' or 'false')
    Include lesion in WM class in hard classification
    flag: --inInclude %s
inMaximum: (an integer (int or long))
    Maximum distance from the interventricular WM boundary to downweight
    the lesion membership to avoid false positives
    flag: --inMaximum %d
inMaximum2: (an integer (int or long))
    Maximum Ventricle Distance
    flag: --inMaximum2 %d
inMaximum3: (an integer (int or long))
    Maximum InterVentricular Distance
    flag: --inMaximum3 %d
inMaximum4: (a float)
    Maximum amount of relative change in the energy function considered
    as the convergence criteria
    flag: --inMaximum4 %f
inMaximum5: (an integer (int or long))
    Maximum iterations
    flag: --inMaximum5 %d
inOutput: ('hard segmentation' or 'hard segmentation+memberships' or
    'cruise inputs' or 'dura removal inputs')
    Output images
    flag: --inOutput %s
inOutput2: ('true' or 'false')

```

```
    Output the hard classification using maximum membership (not
    neceesarily topologically correct)
    flag: --inOutput2 %s
inOutput3: ('true' or 'false')
    Output the estimated inhomogeneity field
    flag: --inOutput3 %s
inSmooting: (a float)
    Controls the effect of neighborhood voxels on the membership
    flag: --inSmooting %f
inT1_MPRAGE: (an existing file name)
    T1_MPRAGE Image
    flag: --inT1_MPRAGE %s
inT1_SPGR: (an existing file name)
    T1_SPGR Image
    flag: --inT1_SPGR %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outCortical: (a boolean or a file name)
    Cortical GM Membership
    flag: --outCortical %s
outFilled: (a boolean or a file name)
    Filled WM Membership
    flag: --outFilled %s
outHard: (a boolean or a file name)
    Hard segmentation
    flag: --outHard %s
outHard2: (a boolean or a file name)
    Hard segmentationfrom memberships
    flag: --outHard2 %s
outInhomogeneity: (a boolean or a file name)
    Inhomogeneity Field
    flag: --outInhomogeneity %s
outLesion: (a boolean or a file name)
    Lesion Segmentation
    flag: --outLesion %s
outMembership: (a boolean or a file name)
    Membership Functions
    flag: --outMembership %s
outSulcal: (a boolean or a file name)
    Sulcal CSF Membership
    flag: --outSulcal %s
outWM: (a boolean or a file name)
    WM Mask
    flag: --outWM %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s
```

Outputs:

```

outCortical: (an existing file name)
    Cortical GM Membership
outFilled: (an existing file name)
    Filled WM Membership
outHard: (an existing file name)
    Hard segmentation
outHard2: (an existing file name)
    Hard segmentationfrom memberships
outInhomogeneity: (an existing file name)
    Inhomogeneity Field
outLesion: (an existing file name)
    Lesion Segmentation
outMembership: (an existing file name)
    Membership Functions
outSulcal: (an existing file name)
    Sulcal CSF Membership
outWM: (an existing file name)
    WM Mask

```

MedicAlgorithmMipavReorient[Link to code](#)

Wraps command ******java edu.jhu.ece.iacI.jist.cli.run edu.jhu.ece.iacI.plugins.utilities.volume.MedicAlgorithmMipavReorient ******

title: Reorient Volume

category: Developer Tools

description: Reorient a volume to a particular anatomical orientation.

version: .alpha

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inInterpolation: ('Nearest Neighbor' or 'Trilinear' or 'Bspline 3rd
    order' or 'Bspline 4th order' or 'Cubic Lagrangian' or 'Quintic
    Lagrangian' or 'Heptic Lagrangian' or 'Windowed Sinc')
    Interpolation
    flag: --inInterpolation %s
inNew: ('Dicom axial' or 'Dicom coronal' or 'Dicom sagittal' or 'User
    defined')
    New image orientation
    flag: --inNew %s
inResolution: ('Unchanged' or 'Finest cubic' or 'Coarsest cubic' or
    'Same as template')
    Resolution

```

```

    flag: --inResolution %s
inSource: (a list of items which are a file name)
    Source
    flag: --inSource %s
inTemplate: (an existing file name)
    Template
    flag: --inTemplate %s
inUser: ('Unknown' or 'Patient Right to Left' or 'Patient Left to
    Right' or 'Patient Posterior to Anterior' or 'Patient Anterior to
    Posterior' or 'Patient Inferior to Superior' or 'Patient Superior
    to Inferior')
    User defined X-axis orientation (image left to right)
    flag: --inUser %s
inUser2: ('Unknown' or 'Patient Right to Left' or 'Patient Left to
    Right' or 'Patient Posterior to Anterior' or 'Patient Anterior to
    Posterior' or 'Patient Inferior to Superior' or 'Patient Superior
    to Inferior')
    User defined Y-axis orientation (image top to bottom)
    flag: --inUser2 %s
inUser3: ('Unknown' or 'Patient Right to Left' or 'Patient Left to
    Right' or 'Patient Posterior to Anterior' or 'Patient Anterior to
    Posterior' or 'Patient Inferior to Superior' or 'Patient Superior
    to Inferior')
    User defined Z-axis orientation (into the screen)
    flag: --inUser3 %s
inUser4: ('Axial' or 'Coronal' or 'Sagittal' or 'Unknown')
    User defined Image Orientation
    flag: --inUser4 %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outReoriented: (a list of items which are a file name)
    Reoriented Volume
    flag: --outReoriented %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

None

MedicAlgorithmN3

[Link to code](#)

Wraps command ****java edu.jhu.ece.iacI.jist.cli.run edu.jhu.ece.iacI.plugins.classification.MedicAlgorithmN3**

title: N3 Correction

category: Developer Tools

description: Non-parametric Intensity Non-uniformity Correction, N3, originally by J.G. Sled.

version: 1.8.R

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inAutomatic: ('true' or 'false')
    If true determines the threshold by histogram analysis. If true a
    VOI cannot be used and the input threshold is ignored.
    flag: --inAutomatic %s
inEnd: (a float)
    Usually 0.01-0.00001, The measure used to terminate the iterations
    is the coefficient of variation of change in field estimates between
    successive iterations.
    flag: --inEnd %f
inField: (a float)
    Characteristic distance over which the field varies. The distance
    between adjacent knots in bspline fitting with at least 4 knots
    going in every dimension. The default in the dialog is one third the
    distance (resolution * extents) of the smallest dimension.
    flag: --inField %f
inInput: (an existing file name)
    Input Volume
    flag: --inInput %s
inKernel: (a float)
    Usually between 0.05-0.50, Width of deconvolution kernel used to
    sharpen the histogram. Larger values give faster convergence while
    smaller values give greater accuracy.
    flag: --inKernel %f
inMaximum: (an integer (int or long))
    Maximum number of Iterations
    flag: --inMaximum %d
inSignal: (a float)
    Default = min + 1, Values at less than threshold are treated as part
    of the background
    flag: --inSignal %f
inSubsample: (a float)
    Usually between 1-32, The factor by which the data is subsampled to
    a lower resolution in estimating the slowly varying non-uniformity
    field. Reduce sampling in the finest sampling direction by the
    shrink factor.
    flag: --inSubsample %f
inWeiner: (a float)
    Usually between 0.0-1.0
    flag: --inWeiner %f
null: (a unicode string)
```

```

    Execution Time
    flag: --null %s
outInhomogeneity: (a boolean or a file name)
    Inhomogeneity Corrected Volume
    flag: --outInhomogeneity %s
outInhomogeneity2: (a boolean or a file name)
    Inhomogeneity Field
    flag: --outInhomogeneity2 %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

```

outInhomogeneity: (an existing file name)
    Inhomogeneity Corrected Volume
outInhomogeneity2: (an existing file name)
    Inhomogeneity Field

```

MedicAlgorithmSPECTRE2010[Link to code](#)

Wraps command ******java edu.jhu.ece.iacI.jist.cli.run edu.jhu.ece.iacI.plugins.segmentation.skull_strip.MedicAlgorithmSPECTRE2******

title: SPECTRE 2010

category: Developer Tools

description: Simple Paradigm for Extra-Cranial Tissue REmoval

Algorithm Version: 1.6 GUI Version: 1.10

A. Carass, M.B. Wheeler, J. Cuzzocreo, P.-L. Bazin, S.S. Bassett, and J.L. Prince, 'A Joint Registration and Segmentation Approach to Skull Stripping', Fourth IEEE International Symposium on Biomedical Imaging (ISBI 2007), Arlington, VA, April 12-15, 2007. A. Carass, J. Cuzzocreo, M.B. Wheeler, P.-L. Bazin, S.M. Resnick, and J.L. Prince, 'Simple paradigm for extra-cerebral tissue removal: Algorithm and analysis', NeuroImage 56(4):1982-1992, 2011.

version: 1.6.R

documentation-url: <http://www.iacI.ece.jhu.edu/>

contributor: Aaron Carass (aaron_carass@jhu.edu) <http://www.iacI.ece.jhu.edu/> Hanlin Wan (hanlin-wan@gmail.com)

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value

```

```

    of class 'str', nipyte default value: {})
    Environment variables
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inApply: ('All' or 'X' or 'Y' or 'Z')
    Apply rotation
    flag: --inApply %s
inAtlas: (an existing file name)
    SPECTRE atlas description file. A text file enumerating atlas files
    and landmarks.
    flag: --inAtlas %s
inBackground: (a float)
    flag: --inBackground %f
inCoarse: (a float)
    Coarse angle increment
    flag: --inCoarse %f
inCost: ('Correlation ratio' or 'Least squares' or 'Normalized cross
    correlation' or 'Normalized mutual information')
    Cost function
    flag: --inCost %s
inDegrees: ('Rigid - 6' or 'Global rescale - 7' or 'Specific rescale
    - 9' or 'Affine - 12')
    Degrees of freedom
    flag: --inDegrees %s
inFind: ('true' or 'false')
    Find Midsagittal Plane
    flag: --inFind %s
inFine: (a float)
    Fine angle increment
    flag: --inFine %f
inImage: ('T1_SPGR' or 'T1_ALT' or 'T1_MPRAGE' or 'T2' or 'FLAIR')
    Set the image modality. MP-RAGE is recommended for most T1 sequence
    images.
    flag: --inImage %s
inInhomogeneity: ('true' or 'false')
    Set to false by default, this parameter will make FANTASM try to do
    inhomogeneity correction during it's iterative cycle.
    flag: --inInhomogeneity %s
inInitial: (an integer (int or long))
    Erosion of the initial mask, which is based on the probability mask
    and the classification., The initial mask is ouput as the d0 volume
    at the conclusion of SPECTRE.
    flag: --inInitial %d
inInitial2: (a float)
    Initial probability threshold
    flag: --inInitial2 %f
inInput: (an existing file name)
    Input volume to be skullstripped.
    flag: --inInput %s
inMMC: (an integer (int or long))
    The size of the dilation step within the Modified Morphological
    Closing.
    flag: --inMMC %d
inMMC2: (an integer (int or long))
    The size of the erosion step within the Modified Morphological
    Closing.
    flag: --inMMC2 %d

```

```
inMaximum: (a float)
    Maximum angle
    flag: --inMaximum %f
inMinimum: (a float)
    Minimum probability threshold
    flag: --inMinimum %f
inMinimum2: (a float)
    Minimum angle
    flag: --inMinimum2 %f
inMultiple: (an integer (int or long))
    Multiple of tolerance to bracket the minimum
    flag: --inMultiple %d
inMultithreading: ('true' or 'false')
    Set to false by default, this parameter controls the multithreaded
    behavior of the linear registration.
    flag: --inMultithreading %s
inNumber: (an integer (int or long))
    Number of iterations
    flag: --inNumber %d
inNumber2: (an integer (int or long))
    Number of minima from Level 8 to test at Level 4
    flag: --inNumber2 %d
inOutput: ('true' or 'false')
    Determines if the output results are transformed back into the space
    of the original input image.
    flag: --inOutput %s
inOutput2: ('true' or 'false')
    Output Plane?
    flag: --inOutput2 %s
inOutput3: ('true' or 'false')
    Output Split-Halves?
    flag: --inOutput3 %s
inOutput4: ('true' or 'false')
    Output Segmentation on Plane?
    flag: --inOutput4 %s
inOutput5: ('Trilinear' or 'Bspline 3rd order' or 'Bspline 4th order'
    or 'Cubic Lagrangian' or 'Quintic Lagrangian' or 'Heptic
    Lagrangian' or 'Windowed sinc' or 'Nearest Neighbor')
    Output interpolation
    flag: --inOutput5 %s
inRegistration: ('Trilinear' or 'Bspline 3rd order' or 'Bspline 4th
    order' or 'Cubic Lagrangian' or 'Quintic Lagrangian' or 'Heptic
    Lagrangian' or 'Windowed sinc')
    Registration interpolation
    flag: --inRegistration %s
inResample: ('true' or 'false')
    Determines if the data is resampled to be isotropic during the
    processing.
    flag: --inResample %s
inRun: ('true' or 'false')
    Run Smooth Brain Mask
    flag: --inRun %s
inSkip: ('true' or 'false')
    Skip multilevel search (Assume images are close to alignment)
    flag: --inSkip %s
inSmoothing: (a float)
    flag: --inSmoothing %f
inSubsample: ('true' or 'false')
```



```

        Subsample image for speed
        flag: --inSubsample %s
inUse: ('true' or 'false')
        Use the max of the min resolutions of the two datasets when
        resampling
        flag: --inUse %s
null: (a unicode string)
        Execution Time
        flag: --null %s
outFANTASM: (a boolean or a file name)
        Tissue classification of of the whole input volume.
        flag: --outFANTASM %s
outMask: (a boolean or a file name)
        Binary Mask of the skullstripped result with just the brain
        flag: --outMask %s
outMidsagittal: (a boolean or a file name)
        Plane dividing the brain hemispheres
        flag: --outMidsagittal %s
outOriginal: (a boolean or a file name)
        If Output in Original Space Flag is true then outputs the original
        input volume. Otherwise outputs the axially reoriented input volume.
        flag: --outOriginal %s
outPrior: (a boolean or a file name)
        Probability prior from the atlas registrations
        flag: --outPrior %s
outSegmentation: (a boolean or a file name)
        2D image showing the tissue classification on the midsagittal plane
        flag: --outSegmentation %s
outSplitHalves: (a boolean or a file name)
        Skullstripped mask of the brain with the hemispheres divided.
        flag: --outSplitHalves %s
outStripped: (a boolean or a file name)
        Skullstripped result of the input volume with just the brain.
        flag: --outStripped %s
outd0: (a boolean or a file name)
        Initial Brainmask
        flag: --outd0 %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
        Set default maximum heap size
        flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
        Set default maximum number of processes.
        flag: -xMaxProcess %d
xPrefExt: ('nrrd')
        Output File Type
        flag: --xPrefExt %s

```

Outputs:

```

outFANTASM: (an existing file name)
        Tissue classification of of the whole input volume.
outMask: (an existing file name)
        Binary Mask of the skullstripped result with just the brain
outMidsagittal: (an existing file name)

```

```

    Plane dividing the brain hemispheres
outOriginal: (an existing file name)
    If Output in Original Space Flag is true then outputs the original
    input volume. Otherwise outputs the axially reoriented input volume.
outPrior: (an existing file name)
    Probability prior from the atlas registrations
outSegmentation: (an existing file name)
    2D image showing the tissue classification on the midsagittal plane
outSplitHalves: (an existing file name)
    Skullstripped mask of the brain with the hemispheres divided.
outStripped: (an existing file name)
    Skullstripped result of the input volume with just the brain.
outd0: (an existing file name)
    Initial Brainmask

```

MedicAlgorithmThresholdToBinaryMask

[Link to code](#)

Wraps command ******`java edu.jhu.ece.iacI.jist.cI.run edu.jhu.ece.iacI.plugins.utilities.volume.MedicAlgorithmThresholdToBinaryM`******

title: Threshold to Binary Mask

category: Developer Tools

description: Given a volume and an intensity range create a binary mask for values within that range.

version: 1.2.RC

documentation-url: <http://www.iacI.ece.jhu.edu/>

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inLabel: (a list of items which are a file name)
    Input volumes
    flag: --inLabel %s
inMaximum: (a float)
    Maximum threshold value.
    flag: --inMaximum %f
inMinimum: (a float)
    Minimum threshold value.
    flag: --inMinimum %f
inUse: ('true' or 'false')
    Use the images max intensity as the max value of the range.
    flag: --inUse %s
null: (a unicode string)
    Execution Time
    flag: --null %s
outBinary: (a list of items which are a file name)
    Binary Mask

```

```

    flag: --outBinary %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s

```

Outputs:

None

RandomVol

[Link to code](#)

Wraps command `**java edu.jhu.ece.iac.jist.cli.run edu.jhu.bme.smile.demo.RandomVol **`

title: Random Volume Generator

category: Developer Tools

description: Generate a random scalar volume.

version: 1.12.RC

documentation-url: <http://www.nitrc.org/projects/jist/>

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inField: ('Uniform' or 'Normal' or 'Exponential')
    Field
    flag: --inField %s
inLambda: (a float)
    Lambda Value for Exponential Distribution
    flag: --inLambda %f
inMaximum: (an integer (int or long))
    Maximum Value
    flag: --inMaximum %d
inMinimum: (an integer (int or long))
    Minimum Value
    flag: --inMinimum %d
inSize: (an integer (int or long))
    Size of Volume in X direction

```

```
    flag: --inSize %d
inSize2: (an integer (int or long))
    Size of Volume in Y direction
    flag: --inSize2 %d
inSize3: (an integer (int or long))
    Size of Volume in Z direction
    flag: --inSize3 %d
inSize4: (an integer (int or long))
    Size of Volume in t direction
    flag: --inSize4 %d
inStandard: (an integer (int or long))
    Standard Deviation for Normal Distribution
    flag: --inStandard %d
null: (a unicode string)
    Execution Time
    flag: --null %s
outRand1: (a boolean or a file name)
    Rand1
    flag: --outRand1 %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    flag: -xDefaultMem %d
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    flag: -xMaxProcess %d
xPrefExt: ('nrrd')
    Output File Type
    flag: --xPrefExt %s
```

Outputs:

```
outRand1: (an existing file name)
    Rand1
```

interfaces.mne.base

WatershedBEM

[Link to code](#)

Wraps command **mne_watershed_bem**

Uses **mne_watershed_bem** to get information from dicom directories

Examples

```
>>> from nipy.interfaces.mne import WatershedBEM
>>> bem = WatershedBEM()
>>> bem.inputs.subject_id = 'subj1'
>>> bem.inputs.subjects_dir = '.'
>>> bem.cmdline
'mne_watershed_bem --overwrite --subject subj1 --volume T1'
>>> bem.run()
```

Inputs:

```
[Mandatory]
subject_id: (a unicode string)
    Subject ID (must have a complete Freesurfer directory)
    flag: --subject %s
subjects_dir: (an existing directory name, nipy default value: )
    Path to Freesurfer subjects directory

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
atlas_mode: (a boolean)
    Use atlas mode for registration (default: no rigid alignment)
    flag: --atlas
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
```

```
Environment variables
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
overwrite: (a boolean, nipyte default value: True)
    Overwrites the existing files
    flag: --overwrite
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
volume: ('T1' or 'aparc+aseg' or 'aseg' or 'brain' or 'orig' or
    'brainmask' or 'ribbon', nipyte default value: T1)
    The volume from the "mri" directory to use (defaults to T1)
    flag: --volume %s
```

Outputs:

```
brain_surface: (an existing file name)
    Brain surface (in Freesurfer format)
cor_files: (a list of items which are an existing file name)
    "COR" format files
fif_file: (an existing file name)
    "fif" format file for EEG processing in MNE
inner_skull_surface: (an existing file name)
    Inner skull surface (in Freesurfer format)
mesh_files: (a list of items which are an existing file name)
    Paths to the output meshes (brain, inner skull, outer skull, outer
    skin)
outer_skin_surface: (an existing file name)
    Outer skin surface (in Freesurfer format)
outer_skull_surface: (an existing file name)
    Outer skull surface (in Freesurfer format)
```

interfaces.mrtrix.convert

MRTrix2TrackVis

[Link to code](#)

Converts MRtrix (.tck) tract files into TrackVis (.trk) format using functions from dipy Example ~~~~~~
>>> import nipype.interfaces.mrtrix as mrt >>> tck2trk = mrt.MRTrix2TrackVis() >>> tck2trk.inputs.in_file =
'dwi_CSD_tracked.tck' >>> tck2trk.inputs.image_file = 'diffusion.nii' >>> tck2trk.run() # doctest: +SKIP
Inputs:

```
[Mandatory]
in_file: (an existing file name)
    The input file for the tracks in MRTrix (.tck) format

[Optional]
image_file: (an existing file name)
    The image the tracks were generated from
matrix_file: (an existing file name)
    A transformation matrix to apply to the tracts after they have been
    generated (from FLIRT - affine transformation from image_file to
    registration_image_file)
out_filename: (a file name, nipype default value: converted.trk)
    The output filename for the tracks in TrackVis (.trk) format
registration_image_file: (an existing file name)
    The final image the tracks should be registered to.
```

Outputs:

```
out_file: (an existing file name)
```

read_mrtrix_header()

[Link to code](#)

read_mrtrix_streamlines()

[Link to code](#)

read_mrtrix_tracks()[Link to code](#)**transform_to_affine()**[Link to code](#)

interfaces.mrtrix.preprocess

DWI2Tensor

[Link to code](#)Wraps command **dwi2tensor**

Converts diffusion-weighted images to tensor images.

Example

```
>>> import nipype.interfaces.mrtrix as mrt
>>> dwi2tensor = mrt.DWI2Tensor()
>>> dwi2tensor.inputs.in_file = 'dwi.mif'
>>> dwi2tensor.inputs.encoding_file = 'encoding.txt'
>>> dwi2tensor.cmdline
'dwi2tensor -grad encoding.txt dwi.mif dwi_tensor.mif'
>>> dwi2tensor.run()
```

Inputs:

```
[Mandatory]
in_file: (a list of items which are an existing file name)
        Diffusion-weighted images
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
debug: (a boolean)
       Display debugging messages.
       flag: -debug, position: 1
encoding_file: (a file name)
               Encoding file supplied as a 4xN text file with each line is in the
               format [ X Y Z b ], where [ X Y Z ] describe the direction of the
               applied gradient, and b gives the b-value in units (1000 s/mm^2).
               See FSL2MRTrix()
               flag: -grad %s, position: 2
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipype default value: {})
         Environment variables
ignore_exception: (a boolean, nipype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
ignore_slice_by_volume: (a list of from 2 to 2 items which are an
                        integer (int or long))
                        Requires two values (i.e. [34 1] for [Slice Volume] Ignores the
                        image slices specified when computing the tensor. Slice here means
```



```

        the z coordinate of the slice to be ignored.
        flag: -ignoreslices %s, position: 2
ignore_volumes: (a list of at least 1 items which are an integer (int
                  or long))
                  Requires two values (i.e. [2 5 6] for [Volumes] Ignores the image
                  volumes specified when computing the tensor.
                  flag: -ignorevolumes %s, position: 2
out_filename: (a file name)
               Output tensor filename
               flag: %s, position: -1
quiet: (a boolean)
       Do not display information messages or progress status.
       flag: -quiet, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

tensor: (an existing file name)
        path/name of output diffusion tensor image

```

Erode[Link to code](#)Wraps command **erode**

Erode (or dilates) a mask (i.e. binary) image

Example

```

>>> import nipy.interfaces.mrtrix as mrt
>>> erode = mrt.Erode()
>>> erode.inputs.in_file = 'mask.mif'
>>> erode.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         Input mask image to be eroded
         flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
debug: (a boolean)
       Display debugging messages.
       flag: -debug, position: 1
dilate: (a boolean)
        Perform dilation rather than erosion
        flag: -dilate, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables

```

```
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
number_of_passes: (an integer (int or long))
    the number of passes (default: 1)
    flag: -npass %s
out_filename: (a file name)
    Output image filename
    flag: %s, position: -1
quiet: (a boolean)
    Do not display information messages or progress status.
    flag: -quiet, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    the output image
```

GenerateWhiteMatterMask

[Link to code](#)Wraps command **gen_WM_mask**

Generates a white matter probability mask from the DW images.

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> genWM = mrt.GenerateWhiteMatterMask()
>>> genWM.inputs.in_file = 'dwi.mif'
>>> genWM.inputs.encoding_file = 'encoding.txt'
>>> genWM.run()
```

Inputs:

```
[Mandatory]
binary_mask: (an existing file name)
    Binary brain mask
    flag: %s, position: -2
encoding_file: (an existing file name)
    Gradient encoding, supplied as a 4xN text file with each line is in
    the format [ X Y Z b ], where [ X Y Z ] describe the direction of
    the applied gradient, and b gives the b-value in units (1000
    s/mm^2). See FSL2MRTrix
    flag: -grad %s, position: 1
in_file: (an existing file name)
    Diffusion-weighted images
    flag: %s, position: -3

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
```

```

        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
noise_level_margin: (a float)
    Specify the width of the margin on either side of the image to be
    used to estimate the noise level (default = 10)
    flag: -margin %s
out_WMPProb_filename: (a file name)
    Output WM probability image filename
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

WMprobabilitymap: (an existing file name)
    WMprobabilitymap

```

MRConvert[Link to code](#)Wraps command **mrconvert**

Perform conversion between different file types and optionally extract a subset of the input image.

If used correctly, this program can be a very useful workhorse. In addition to converting images between different formats, it can be used to extract specific studies from a data set, extract a specific region of interest, flip the images, or to scale the intensity of the images.

Example

```

>>> import nipy.interfaces.mrtrix as mrt
>>> mrconvert = mrt.MRConvert()
>>> mrconvert.inputs.in_file = 'dwi_FA.mif'
>>> mrconvert.inputs.out_filename = 'dwi_FA.nii'
>>> mrconvert.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    voxel-order data filename
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
extension: ('mif' or 'nii' or 'float' or 'char' or 'short' or 'int'
    or 'long' or 'double', nipy default value: mif)

```

```

    "i.e. Bfloat". Can be "char", "short", "int", "long", "float" or
    "double"
extract_at_axis: (1 or 2 or 3)
    "Extract data only at the coordinates specified. This option
    specifies the Axis. Must be used in conjunction with
    extract_at_coordinate.
    flag: -coord %s, position: 1
extract_at_coordinate: (a list of from 1 to 3 items which are a
    float)
    "Extract data only at the coordinates specified. This option
    specifies the coordinates. Must be used in conjunction with
    extract_at_axis. Three comma-separated numbers giving the size of
    each voxel in mm.
    flag: %s, position: 2
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
layout: ('nii' or 'float' or 'char' or 'short' or 'int' or 'long' or
    'double')
    specify the layout of the data in memory. The actual layout produced
    will depend on whether the output image format can support it.
    flag: -output %s, position: 2
offset_bias: (a float)
    Apply offset to the intensity values.
    flag: -scale %d, position: 3
out_filename: (a file name)
    Output filename
    flag: %s, position: -1
output_datatype: ('nii' or 'float' or 'char' or 'short' or 'int' or
    'long' or 'double')
    "i.e. Bfloat". Can be "char", "short", "int", "long", "float" or
    "double"
    flag: -output %s, position: 2
prs: (a boolean)
    Assume that the DW gradients are specified in the PRS frame (Siemens
    DICOM only).
    flag: -prs, position: 3
replace_NaN_with_zero: (a boolean)
    Replace all NaN values with zero.
    flag: -zero, position: 3
resample: (a float)
    Apply scaling to the intensity values.
    flag: -scale %d, position: 3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
voxel_dims: (a list of from 3 to 3 items which are a float)
    Three comma-separated numbers giving the size of each voxel in mm.
    flag: -vox %s, position: 3

```

Outputs:

```

converted: (an existing file name)
    path/name of 4D volume in voxel order

```

MRMultiply

[Link to code](#)

Wraps command **mrmult**

Multiplies two images.

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> MRmult = mrt.MRMultiply()
>>> MRmult.inputs.in_files = ['dwi.mif', 'dwi_WMPProb.mif']
>>> MRmult.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
    Input images to be multiplied
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debug: (a boolean)
    Display debugging messages.
    flag: -debug, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_filename: (a file name)
    Output image filename
    flag: %s, position: -1
quiet: (a boolean)
    Do not display information messages or progress status.
    flag: -quiet, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    the output image of the multiplication
```

MRTransform

[Link to code](#)

Wraps command **mrtransform**

Apply spatial transformations or reslice images

Example

```
>>> MRxform = MRTransform()
>>> MRxform.inputs.in_files = 'anat_coreg.mif'
>>> MRxform.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
    Input images to be transformed
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debug: (a boolean)
    Display debugging messages.
    flag: -debug, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
flip_x: (a boolean)
    assume the transform is supplied assuming a coordinate system with
    the x-axis reversed relative to the MRtrix convention (i.e. x
    increases from right to left). This is required to handle transform
    matrices produced by FSL's FLIRT command. This is only used in
    conjunction with the -reference option.
    flag: -flipx, position: 1
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert: (a boolean)
    Invert the specified transform before using it
    flag: -inverse, position: 1
out_filename: (a file name)
    Output image
    flag: %s, position: -1
quiet: (a boolean)
    Do not display information messages or progress status.
    flag: -quiet, position: 1
reference_image: (an existing file name)
    in case the transform supplied maps from the input image onto a
    reference image, use this option to specify the reference. Note that
    this implicitly sets the -replace option.
    flag: -reference %s, position: 1
replace_transform: (a boolean)
    replace the current transform by that specified, rather than
    applying it to the current transform
    flag: -replace, position: 1
template_image: (an existing file name)
    Reslice the input image to match the specified template image.
    flag: -template %s, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

```
transformation_file: (an existing file name)
    The transform to apply, in the form of a 4x4 ascii file.
    flag: -transform %s, position: 1
```

Outputs:

```
out_file: (an existing file name)
    the output image of the transformation
```

MRTrixViewer[Link to code](#)Wraps command **mrview**

Loads the input images in the MRTrix Viewer.

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> MRview = mrt.MRTrixViewer()
>>> MRview.inputs.in_files = 'dwi.mif'
>>> MRview.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
    Input images to be viewed
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debug: (a boolean)
    Display debugging messages.
    flag: -debug, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
quiet: (a boolean)
    Do not display information messages or progress status.
    flag: -quiet, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:**None****MedianFilter3D**[Link to code](#)

Wraps command **median3D**

Smooth images using a 3x3x3 median filter.

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> median3d = mrt.MedianFilter3D()
>>> median3d.inputs.in_file = 'mask.mif'
>>> median3d.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Input images to be smoothed
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
debug: (a boolean)
       Display debugging messages.
       flag: -debug, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_filename: (a file name)
              Output image filename
              flag: %s, position: -1
quiet: (a boolean)
       Do not display information messages or progress status.
       flag: -quiet, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          the output image
```

Tensor2ApparentDiffusion

[Link to code](#)

Wraps command **tensor2ADC**

Generates a map of the apparent diffusion coefficient (ADC) in each voxel

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> tensor2ADC = mrt.Tensor2ApparentDiffusion()
```



```
>>> tensor2ADC.inputs.in_file = 'dwi_tensor.mif'
>>> tensor2ADC.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Diffusion tensor image
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
debug: (a boolean)
       Display debugging messages.
       flag: -debug, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_filename: (a file name)
              Output Fractional Anisotropy filename
              flag: %s, position: -1
quiet: (a boolean)
       Do not display information messages or progress status.
       flag: -quiet, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
ADC: (an existing file name)
     the output image of the major eigenvectors of the diffusion tensor
     image.
```

Tensor2FractionalAnisotropy

[Link to code](#)Wraps command **tensor2FA**

Generates a map of the fractional anisotropy in each voxel.

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> tensor2FA = mrt.Tensor2FractionalAnisotropy()
>>> tensor2FA.inputs.in_file = 'dwi_tensor.mif'
>>> tensor2FA.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
```

```

        Diffusion tensor image
        flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debug: (a boolean)
    Display debugging messages.
    flag: -debug, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_filename: (a file name)
    Output Fractional Anisotropy filename
    flag: %s, position: -1
quiet: (a boolean)
    Do not display information messages or progress status.
    flag: -quiet, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

FA: (an existing file name)
    the output image of the major eigenvectors of the diffusion tensor
    image.

```

Tensor2Vector

[Link to code](#)Wraps command **tensor2vector**

Generates a map of the major eigenvectors of the tensors in each voxel.

Example

```

>>> import nipy.interfaces.mrtrix as mrt
>>> tensor2vector = mrt.Tensor2Vector()
>>> tensor2vector.inputs.in_file = 'dwi_tensor.mif'
>>> tensor2vector.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    Diffusion tensor image
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command

```

```

    flag: %s
debug: (a boolean)
    Display debugging messages.
    flag: -debug, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_filename: (a file name)
    Output vector filename
    flag: %s, position: -1
quiet: (a boolean)
    Do not display information messages or progress status.
    flag: -quiet, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

vector: (an existing file name)
    the output image of the major eigenvectors of the diffusion tensor
    image.

```

Threshold[Link to code](#)Wraps command **threshold**

Create bitwise image by thresholding image intensity.

By default, the threshold level is determined using a histogram analysis to cut out the background. Otherwise, the threshold intensity can be specified using command line options. Note that only the first study is used for thresholding.

Example

```

>>> import nipy.interfaces.mrtrix as mrt
>>> thresh = mrt.Threshold()
>>> thresh.inputs.in_file = 'wm_mask.mif'
>>> thresh.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    The input image to be thresholded
    flag: %s, position: -2

[Optional]
absolute_threshold_value: (a float)
    Specify threshold value as absolute intensity.
    flag: -abs %s
args: (a unicode string)
    Additional parameters to the command

```

```

    flag: %s
debug: (a boolean)
    Display debugging messages.
    flag: -debug, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
invert: (a boolean)
    Invert output binary mask
    flag: -invert, position: 1
out_filename: (a file name)
    The output binary image mask.
    flag: %s, position: -1
percentage_threshold_value: (a float)
    Specify threshold value as a percentage of the peak intensity in the
    input image.
    flag: -percent %s
quiet: (a boolean)
    Do not display information messages or progress status.
    flag: -quiet, position: 1
replace_zeros_with_NaN: (a boolean)
    Replace all zero values with NaN
    flag: -nan, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    The output binary image mask.

```

interfaces.mrtrix.tensors

ConstrainedSphericalDeconvolution

[Link to code](#)Wraps command **csdeconv**

Perform non-negativity constrained spherical deconvolution.

Note that this program makes use of implied symmetries in the diffusion profile. First, the fact the signal attenuation profile is real implies that it has conjugate symmetry, i.e. $Y(l,-m) = Y(l,m)^*$ (where $*$ denotes the complex conjugate). Second, the diffusion profile should be antipodally symmetric (i.e. $S(x) = S(-x)$), implying that all odd l components should be zero. Therefore, this program only computes the even elements. Note that the spherical harmonics equations used here differ slightly from those conventionally used, in that the $(-1)^m$ factor has been omitted. This should be taken into account in all subsequent calculations. Each volume in the output image corresponds to a different spherical harmonic component, according to the following convention:

- [0] $Y(0,0)$
- [1] $\text{Im} \{Y(2,2)\}$
- [2] $\text{Im} \{Y(2,1)\}$
- [3] $Y(2,0)$
- [4] $\text{Re} \{Y(2,1)\}$

- [5] Re {Y(2,2)}
- [6] Im {Y(4,4)}
- [7] Im {Y(4,3)}

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> csdeconv = mrt.ConstrainedSphericalDeconvolution()
>>> csdeconv.inputs.in_file = 'dwi.mif'
>>> csdeconv.inputs.encoding_file = 'encoding.txt'
>>> csdeconv.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         diffusion-weighted image
         flag: %s, position: -3
response_file: (an existing file name)
               the diffusion-weighted signal response function for a single fibre
               population (see EstimateResponse)
               flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
debug: (a boolean)
       Display debugging messages.
       flag: -debug
directions_file: (an existing file name)
                 a text file containing the [ el az ] pairs for the directions:
                 Specify the directions over which to apply the non-negativity
                 constraint (by default, the built-in 300 direction set is used)
                 flag: -directions %s, position: -2
encoding_file: (an existing file name)
               Gradient encoding, supplied as a 4xN text file with each line is in
               the format [ X Y Z b ], where [ X Y Z ] describe the direction of
               the applied gradient, and b gives the b-value in units (1000
               s/mm^2). See FSL2MRTrix
               flag: -grad %s, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
filter_file: (an existing file name)
            a text file containing the filtering coefficients for each even
            harmonic order.the linear frequency filtering parameters used for
            the initial linear spherical deconvolution step (default = [ 1 1 1 0
            0 ]).
            flag: -filter %s, position: -2
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
iterations: (an integer (int or long))
            the maximum number of iterations to perform for each voxel (default
            = 50)
            flag: -niter %s
```

```

lambda_value: (a float)
    the regularisation parameter lambda that controls the strength of
    the constraint (default = 1.0).
    flag: -lambda %s
mask_image: (an existing file name)
    only perform computation within the specified binary brain mask
    image
    flag: -mask %s, position: 2
maximum_harmonic_order: (an integer (int or long))
    set the maximum harmonic order for the output series. By default,
    the program will use the highest possible lmax given the number of
    diffusion-weighted images.
    flag: -lmax %s
normalise: (a boolean)
    normalise the DW signal to the b=0 image
    flag: -normalise, position: 3
out_filename: (a file name)
    Output filename
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold_value: (a float)
    the threshold below which the amplitude of the FOD is assumed to be
    zero, expressed as a fraction of the mean value of the initial FOD
    (default = 0.1)
    flag: -threshold %s

```

Outputs:

```

spherical_harmonics_image: (an existing file name)
    Spherical harmonics image

```

DWI2SphericalHarmonicsImage[Link to code](#)**Wraps command `dwi2SH`**

Convert base diffusion-weighted images to their spherical harmonic representation.

This program outputs the spherical harmonic decomposition for the set measured signal attenuations. The signal attenuations are calculated by identifying the b-zero images from the diffusion encoding supplied (i.e. those with zero as the b-value), and dividing the remaining signals by the mean b-zero signal intensity. The spherical harmonic decomposition is then calculated by least-squares linear fitting. Note that this program makes use of implied symmetries in the diffusion profile.

First, the fact the signal attenuation profile is real implies that it has conjugate symmetry, i.e. $Y(l, -m) = Y(l, m)^*$ (where $*$ denotes the complex conjugate). Second, the diffusion profile should be antipodally symmetric (i.e. $S(x) = S(-x)$), implying that all odd l components should be zero. Therefore, this program only computes the even elements.

Note that the spherical harmonics equations used here differ slightly from those conventionally used, in that the $(-1)^m$ factor has been omitted. This should be taken into account in all subsequent calculations.

Each volume in the output image corresponds to a different spherical harmonic component, according to the following convention:

- [0] $Y(0,0)$
- [1] $\text{Im} \{Y(2,2)\}$
- [2] $\text{Im} \{Y(2,1)\}$
- [3] $Y(2,0)$

- [4] Re {Y(2,1)}
- [5] Re {Y(2,2)}
- [6] Im {Y(4,4)}
- [7] Im {Y(4,3)}

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> dwi2SH = mrt.DWI2SphericalHarmonicsImage()
>>> dwi2SH.inputs.in_file = 'diffusion.nii'
>>> dwi2SH.inputs.encoding_file = 'encoding.txt'
>>> dwi2SH.run()
```

Inputs:

```
[Mandatory]
encoding_file: (an existing file name)
    Gradient encoding, supplied as a 4xN text file with each line is in
    the format [ X Y Z b ], where [ X Y Z ] describe the direction of
    the applied gradient, and b gives the b-value in units (1000
    s/mm^2). See FSL2MRtrix
    flag: -grad %s, position: 1
in_file: (an existing file name)
    Diffusion-weighted images
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
maximum_harmonic_order: (a float)
    set the maximum harmonic order for the output series. By default,
    the program will use the highest possible lmax given the number of
    diffusion-weighted images.
    flag: -lmax %s
normalise: (a boolean)
    normalise the DW signal to the b=0 image
    flag: -normalise, position: 3
out_filename: (a file name)
    Output filename
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
spherical_harmonics_image: (an existing file name)
    Spherical harmonics image
```

Directions2Amplitude

[Link to code](#)

Wraps command **dir2amp**

convert directions image to amplitudes

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> amplitudes = mrt.Directions2Amplitude()
>>> amplitudes.inputs.in_file = 'peak_directions.mif'
>>> amplitudes.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    the input directions image. Each volume corresponds to the x, y & z
    component of each direction vector in turn.
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
display_debug: (a boolean)
    Display debugging messages.
    flag: -debug
display_info: (a boolean)
    Display information messages.
    flag: -info
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
num_peaks: (an integer (int or long))
    the number of peaks to extract (default is 3)
    flag: -num %s
out_file: (a file name)
    the output amplitudes image
    flag: %s, position: -1
peak_directions: (a list of from 2 to 2 items which are a float)
    phi theta. the direction of a peak to estimate. The algorithm will
    attempt to find the same number of peaks as have been specified
    using this option phi: the azimuthal angle of the direction (in
    degrees). theta: the elevation angle of the direction (in degrees,
    from the vertical z-axis)
    flag: -direction %s
peaks_image: (an existing file name)
    the program will try to find the peaks that most closely match those
    in the image provided
    flag: -peaks %s
quiet_display: (a boolean)
    do not display information messages or progress status.
    flag: -quiet
```



```
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    amplitudes image
```

EstimateResponseForSH

[Link to code](#)Wraps command **estimate_response**

Estimates the fibre response function for use in spherical deconvolution.

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> estresp = mrt.EstimateResponseForSH()
>>> estresp.inputs.in_file = 'dwi.mif'
>>> estresp.inputs.mask_image = 'dwi_WMPProb.mif'
>>> estresp.inputs.encoding_file = 'encoding.txt'
>>> estresp.run()
```

Inputs:

```
[Mandatory]
encoding_file: (an existing file name)
    Gradient encoding, supplied as a 4xN text file with each line is in
    the format [ X Y Z b ], where [ X Y Z ] describe the direction of
    the applied gradient, and b gives the b-value in units (1000
    s/mm^2). See FSL2MRTrix
    flag: -grad %s, position: 1
in_file: (an existing file name)
    Diffusion-weighted images
    flag: %s, position: -3
mask_image: (an existing file name)
    only perform computation within the specified binary brain mask
    image
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debug: (a boolean)
    Display debugging messages.
    flag: -debug
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
maximum_harmonic_order: (an integer (int or long))
```

```

        set the maximum harmonic order for the output series. By default,
        the program will use the highest possible lmax given the number of
        diffusion-weighted images.
        flag: -lmax %s
normalise: (a boolean)
        normalise the DW signal to the b=0 image
        flag: -normalise
out_filename: (a file name)
        Output filename
        flag: %s, position: -1
quiet: (a boolean)
        Do not display information messages or progress status.
        flag: -quiet
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

response: (an existing file name)
        Spherical harmonics image

```

FindShPeaks[Link to code](#)**Wraps command `find_SH_peaks`**

identify the orientations of the N largest peaks of a SH profile

Example

```

>>> import nipy.interfaces.mrtrix as mrt
>>> shpeaks = mrt.FindShPeaks()
>>> shpeaks.inputs.in_file = 'csd.mif'
>>> shpeaks.inputs.directions_file = 'dirs.txt'
>>> shpeaks.inputs.num_peaks = 2
>>> shpeaks.run()

```

Inputs:

```

[Mandatory]
directions_file: (an existing file name)
        the set of directions to use as seeds for the peak finding
        flag: %s, position: -2
in_file: (an existing file name)
        the input image of SH coefficients.
        flag: %s, position: -3

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
display_debug: (a boolean)
        Display debugging messages.
        flag: -debug
display_info: (a boolean)
        Display information messages.

```

```

    flag: -info
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
num_peaks: (an integer (int or long))
    the number of peaks to extract (default is 3)
    flag: -num %s
out_file: (a file name)
    the output image. Each volume corresponds to the x, y & z component
    of each peak direction vector in turn
    flag: %s, position: -1
peak_directions: (a list of from 2 to 2 items which are a float)
    phi theta. the direction of a peak to estimate. The algorithm will
    attempt to find the same number of peaks as have been specified
    using this option phi: the azimuthal angle of the direction (in
    degrees). theta: the elevation angle of the direction (in degrees,
    from the vertical z-axis)
    flag: -direction %s
peak_threshold: (a float)
    only peak amplitudes greater than the threshold will be considered
    flag: -threshold %s
peaks_image: (an existing file name)
    the program will try to find the peaks that most closely match those
    in the image provided
    flag: -peaks %s
quiet_display: (a boolean)
    do not display information messages or progress status.
    flag: -quiet
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    Peak directions image

```

GenerateDirections[Link to code](#)Wraps command **gendir**

generate a set of directions evenly distributed over a hemisphere.

Example

```

>>> import nipy.interfaces.mrtrix as mrt
>>> gendir = mrt.GenerateDirections()
>>> gendir.inputs.num_dirs = 300
>>> gendir.run()

```

Inputs:

```
[Mandatory]
num_dirs: (an integer (int or long))
    the number of directions to generate.
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
display_debug: (a boolean)
    Display debugging messages.
    flag: -debug
display_info: (a boolean)
    Display information messages.
    flag: -info
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
niter: (an integer (int or long))
    specify the maximum number of iterations to perform.
    flag: -niter %s
out_file: (a file name)
    the text file to write the directions to, as [ az el ] pairs.
    flag: %s, position: -1
power: (a float)
    specify exponent to use for repulsion power law.
    flag: -power %s
quiet_display: (a boolean)
    do not display information messages or progress status.
    flag: -quiet
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    directions file
```

concat_files()[Link to code](#)**interfaces.mrtrix.tracking****DiffusionTensorStreamlineTrack**[Link to code](#)Wraps command **streamtrack**

Specialized interface to StreamlineTrack. This interface is used for streamline tracking from diffusion tensor data, and calls the MRtrix function 'streamtrack' with the option 'DT_STREAM'

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> dtstrack = mrt.DiffusionTensorStreamlineTrack()
>>> dtstrack.inputs.in_file = 'data.Bfloat'
>>> dtstrack.inputs.seed_file = 'seed_mask.nii'
>>> dtstrack.run()
```

Inputs:

[Mandatory]

gradient_encoding_file: (an existing file name)
 Gradient encoding, supplied as a 4xN text file with each line is in the format [X Y Z b], where [X Y Z] describe the direction of the applied gradient, and b gives the b-value in units (1000 s/mm²). See FSL2MRTrix
 flag: -grad %s, position: -2

in_file: (an existing file name)
 the image containing the source data. The type of data required depends on the type of tracking as set in the preceding argument. For DT methods, the base DWI are needed. For SD methods, the SH harmonic coefficients of the FOD are needed.
 flag: %s, position: -2

[Optional]

args: (a unicode string)
 Additional parameters to the command
 flag: %s

cutoff_value: (a float)
 Set the FA or FOD amplitude cutoff for terminating tracks (default is 0.1).
 flag: -cutoff %s

desired_number_of_tracks: (an integer (int or long))
 Sets the desired number of tracks. The program will continue to generate tracks until this number of tracks have been selected and written to the output file (default is 100 for *_STREAM methods, 1000 for *_PROB methods).
 flag: -number %d

do_not_precompute: (a boolean)
 Turns off precomputation of the legendre polynomial values. Warning: this will slow down the algorithm by a factor of approximately 4.
 flag: -noprecomputed

environ: (a dictionary with keys which are a bytes or None or a value of class 'str' and with values which are a bytes or None or a value of class 'str', nipy default value: {})
 Environment variables

exclude_file: (an existing file name)
 exclusion file
 flag: -exclude %s
 mutually_exclusive: exclude_file, exclude_spec

exclude_spec: (a list of from 4 to 4 items which are a float)
 exclusion specification in mm and radius (x y z r)
 flag: -exclude %s, position: 2
 mutually_exclusive: exclude_file, exclude_spec

ignore_exception: (a boolean, nipy default value: False)
 Print an error message instead of throwing an exception in case the interface fails to run

include_file: (an existing file name)
 inclusion file

```

    flag: -include %s
    mutually_exclusive: include_file, include_spec
include_spec: (a list of from 4 to 4 items which are a float)
    inclusion specification in mm and radius (x y z r)
    flag: -include %s, position: 2
    mutually_exclusive: include_file, include_spec
initial_cutoff_value: (a float)
    Sets the minimum FA or FOD amplitude for initiating tracks (default
    is twice the normal cutoff).
    flag: -initcutoff %s
initial_direction: (a list of from 2 to 2 items which are an integer
    (int or long))
    Specify the initial tracking direction as a vector
    flag: -initdirection %s
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipy default
    value: DT_STREAM)
    input model type
    flag: %s, position: -3
mask_file: (an existing file name)
    mask file. Only tracks within mask.
    flag: -mask %s
    mutually_exclusive: mask_file, mask_spec
mask_spec: (a list of from 4 to 4 items which are a float)
    Mask specification in mm and radius (x y z r). Tracks will be
    terminated when they leave the ROI.
    flag: -mask %s, position: 2
    mutually_exclusive: mask_file, mask_spec
maximum_number_of_tracks: (an integer (int or long))
    Sets the maximum number of tracks to generate. The program will not
    generate more tracks than this number, even if the desired number of
    tracks hasn't yet been reached (default is 100 x number).
    flag: -maxnum %d
maximum_tract_length: (a float)
    Sets the maximum length of any track in millimeters (default is 200
    mm).
    flag: -length %s
minimum_radius_of_curvature: (a float)
    Set the minimum radius of curvature (default is 2 mm for DT_STREAM,
    0 for SD_STREAM, 1 mm for SD_PROB and DT_PROB)
    flag: -curvature %s
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    flag: -minlength %s
no_mask_interpolation: (a boolean)
    Turns off trilinear interpolation of mask images.
    flag: -nomaskinterp
out_file: (a file name)
    output data file
    flag: %s, position: -1
seed_file: (an existing file name)
    seed file
    flag: -seed %s
    mutually_exclusive: seed_file, seed_spec
seed_spec: (a list of from 4 to 4 items which are a float)
    seed specification in mm and radius (x y z r)
    flag: -seed %s, position: 2
    mutually_exclusive: seed_file, seed_spec

```

```

step_size: (a float)
    Set the step size of the algorithm in mm (default is 0.2).
    flag: -step %s
stop: (a boolean)
    stop track as soon as it enters any of the include regions.
    flag: -stop
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
unidirectional: (a boolean)
    Track from the seed point in one direction only (default is to track
    in both directions).
    flag: -unidirectional

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracks

```

FilterTracks[Link to code](#)Wraps command **filter_tracks**

Use regions-of-interest to select a subset of tracks from a given MRtrix track file.

Example

```

>>> import nipype.interfaces.mrtrix as mrt
>>> filt = mrt.FilterTracks()
>>> filt.inputs.in_file = 'tracks.tck'
>>> filt.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input tracks to be filtered
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debug: (a boolean)
    Display debugging messages.
    flag: -debug, position: 1
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
exclude_file: (an existing file name)
    exclusion file
    flag: -exclude %s
    mutually_exclusive: exclude_file, exclude_spec
exclude_spec: (a list of from 4 to 4 items which are a float)
    exclusion specification in mm and radius (x y z r)

```

```

    flag: -exclude %s, position: 2
    mutually_exclusive: exclude_file, exclude_spec
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
include_file: (an existing file name)
    inclusion file
    flag: -include %s
    mutually_exclusive: include_file, include_spec
include_spec: (a list of from 4 to 4 items which are a float)
    inclusion specification in mm and radius (x y z r)
    flag: -include %s, position: 2
    mutually_exclusive: include_file, include_spec
invert: (a boolean)
    invert the matching process, so that tracks that would otherwise have
    been included are now excluded and vice-versa.
    flag: -invert
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    flag: -minlength %s
no_mask_interpolation: (a boolean)
    Turns off trilinear interpolation of mask images.
    flag: -nomaskinterp
out_file: (a file name)
    Output filtered track filename
    flag: %s, position: -1
quiet: (a boolean)
    Do not display information messages or progress status.
    flag: -quiet, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    the output filtered tracks

```

ProbabilisticSphericallyDeconvolutedStreamlineTrack[Link to code](#)Wraps command **streamtrack**

Performs probabilistic tracking using spherically deconvolved data

Specialized interface to StreamlineTrack. This interface is used for probabilistic tracking from spherically deconvolved data, and calls the MRtrix function 'streamtrack' with the option 'SD_PROB'

Example

```

>>> import nipy.interfaces.mrtrix as mrt
>>> sdprobtrack = mrt.ProbabilisticSphericallyDeconvolutedStreamlineTrack()
>>> sdprobtrack.inputs.in_file = 'data.Bfloat'
>>> sdprobtrack.inputs.seed_file = 'seed_mask.nii'
>>> sdprobtrack.run()

```

Inputs:


```

[Mandatory]
in_file: (an existing file name)
    the image containing the source data.The type of data required
    depends on the type of tracking as set in the preceeding argument.
    For DT methods, the base DWI are needed. For SD methods, the SH
    harmonic coefficients of the FOD are needed.
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cutoff_value: (a float)
    Set the FA or FOD amplitude cutoff for terminating tracks (default
    is 0.1).
    flag: -cutoff %s
desired_number_of_tracks: (an integer (int or long))
    Sets the desired number of tracks.The program will continue to
    generate tracks until this number of tracks have been selected and
    written to the output file(default is 100 for *_STREAM methods, 1000
    for *_PROB methods).
    flag: -number %d
do_not_precompute: (a boolean)
    Turns off precomputation of the legendre polynomial values. Warning:
    this will slow down the algorithm by a factor of approximately 4.
    flag: -noprecomputed
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
exclude_file: (an existing file name)
    exclusion file
    flag: -exclude %s
    mutually_exclusive: exclude_file, exclude_spec
exclude_spec: (a list of from 4 to 4 items which are a float)
    exclusion specification in mm and radius (x y z r)
    flag: -exclude %s, position: 2
    mutually_exclusive: exclude_file, exclude_spec
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
include_file: (an existing file name)
    inclusion file
    flag: -include %s
    mutually_exclusive: include_file, include_spec
include_spec: (a list of from 4 to 4 items which are a float)
    inclusion specification in mm and radius (x y z r)
    flag: -include %s, position: 2
    mutually_exclusive: include_file, include_spec
initial_cutoff_value: (a float)
    Sets the minimum FA or FOD amplitude for initiating tracks (default
    is twice the normal cutoff).
    flag: -initcutoff %s
initial_direction: (a list of from 2 to 2 items which are an integer
    (int or long))
    Specify the initial tracking direction as a vector
    flag: -initdirection %s
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipy default

```

```

    value: DT_STREAM)
    input model type
    flag: %s, position: -3
mask_file: (an existing file name)
    mask file. Only tracks within mask.
    flag: -mask %s
    mutually_exclusive: mask_file, mask_spec
mask_spec: (a list of from 4 to 4 items which are a float)
    Mask specification in mm and radius (x y z r). Tracks will be
    terminated when they leave the ROI.
    flag: -mask %s, position: 2
    mutually_exclusive: mask_file, mask_spec
maximum_number_of_tracks: (an integer (int or long))
    Sets the maximum number of tracks to generate.The program will not
    generate more tracks than this number, even if the desired number of
    tracks hasn't yet been reached(default is 100 x number).
    flag: -maxnum %d
maximum_number_of_trials: (an integer (int or long))
    Set the maximum number of sampling trials at each point (only used
    for probabilistic tracking).
    flag: -trials %s
maximum_tract_length: (a float)
    Sets the maximum length of any track in millimeters (default is 200
    mm).
    flag: -length %s
minimum_radius_of_curvature: (a float)
    Set the minimum radius of curvature (default is 2 mm for DT_STREAM,
    0 for SD_STREAM, 1 mm for SD_PROB and DT_PROB)
    flag: -curvature %s
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    flag: -minlength %s
no_mask_interpolation: (a boolean)
    Turns off trilinear interpolation of mask images.
    flag: -nomaskinterp
out_file: (a file name)
    output data file
    flag: %s, position: -1
seed_file: (an existing file name)
    seed file
    flag: -seed %s
    mutually_exclusive: seed_file, seed_spec
seed_spec: (a list of from 4 to 4 items which are a float)
    seed specification in mm and radius (x y z r)
    flag: -seed %s, position: 2
    mutually_exclusive: seed_file, seed_spec
step_size: (a float)
    Set the step size of the algorithm in mm (default is 0.2).
    flag: -step %s
stop: (a boolean)
    stop track as soon as it enters any of the include regions.
    flag: -stop
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
unidirectional: (a boolean)

```

```
Track from the seed point in one direction only (default is to track
in both directions).
flag: -unidirectional
```

Outputs:

```
tracked: (an existing file name)
        output file containing reconstructed tracts
```

SphericallyDeconvolutedStreamlineTrack[Link to code](#)Wraps command **streamtrack**

Performs streamline tracking using spherically deconvolved data

Specialized interface to StreamlineTrack. This interface is used for streamline tracking from spherically deconvolved data, and calls the MRtrix function 'streamtrack' with the option 'SD_STREAM'

Example

```
>>> import nipy.interfaces.mrtrix as mrt
>>> sdtrack = mrt.SphericallyDeconvolutedStreamlineTrack()
>>> sdtrack.inputs.in_file = 'data.Bfloat'
>>> sdtrack.inputs.seed_file = 'seed_mask.nii'
>>> sdtrack.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        the image containing the source data.The type of data required
        depends on the type of tracking as set in the preceeding argument.
        For DT methods, the base DWI are needed. For SD methods, the SH
        harmonic coefficients of the FOD are needed.
        flag: %s, position: -2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
cutoff_value: (a float)
        Set the FA or FOD amplitude cutoff for terminating tracks (default
        is 0.1).
        flag: -cutoff %s
desired_number_of_tracks: (an integer (int or long))
        Sets the desired number of tracks.The program will continue to
        generate tracks until this number of tracks have been selected and
        written to the output file(default is 100 for *_STREAM methods, 1000
        for *_PROB methods).
        flag: -number %d
do_not_precompute: (a boolean)
        Turns off precomputation of the legendre polynomial values. Warning:
        this will slow down the algorithm by a factor of approximately 4.
        flag: -noprocomputed
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
```

```

exclude_file: (an existing file name)
    exclusion file
    flag: -exclude %s
    mutually_exclusive: exclude_file, exclude_spec
exclude_spec: (a list of from 4 to 4 items which are a float)
    exclusion specification in mm and radius (x y z r)
    flag: -exclude %s, position: 2
    mutually_exclusive: exclude_file, exclude_spec
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
include_file: (an existing file name)
    inclusion file
    flag: -include %s
    mutually_exclusive: include_file, include_spec
include_spec: (a list of from 4 to 4 items which are a float)
    inclusion specification in mm and radius (x y z r)
    flag: -include %s, position: 2
    mutually_exclusive: include_file, include_spec
initial_cutoff_value: (a float)
    Sets the minimum FA or FOD amplitude for initiating tracks (default
    is twice the normal cutoff).
    flag: -initcutoff %s
initial_direction: (a list of from 2 to 2 items which are an integer
    (int or long))
    Specify the initial tracking direction as a vector
    flag: -initdirection %s
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipy default
    value: DT_STREAM)
    input model type
    flag: %s, position: -3
mask_file: (an existing file name)
    mask file. Only tracks within mask.
    flag: -mask %s
    mutually_exclusive: mask_file, mask_spec
mask_spec: (a list of from 4 to 4 items which are a float)
    Mask specification in mm and radius (x y z r). Tracks will be
    terminated when they leave the ROI.
    flag: -mask %s, position: 2
    mutually_exclusive: mask_file, mask_spec
maximum_number_of_tracks: (an integer (int or long))
    Sets the maximum number of tracks to generate. The program will not
    generate more tracks than this number, even if the desired number of
    tracks hasn't yet been reached (default is 100 x number).
    flag: -maxnum %d
maximum_tract_length: (a float)
    Sets the maximum length of any track in millimeters (default is 200
    mm).
    flag: -length %s
minimum_radius_of_curvature: (a float)
    Set the minimum radius of curvature (default is 2 mm for DT_STREAM,
    0 for SD_STREAM, 1 mm for SD_PROB and DT_PROB)
    flag: -curvature %s
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    flag: -minlength %s
no_mask_interpolation: (a boolean)

```

```

        Turns off trilinear interpolation of mask images.
        flag: -nomaskinterp
out_file: (a file name)
        output data file
        flag: %s, position: -1
seed_file: (an existing file name)
        seed file
        flag: -seed %s
        mutually_exclusive: seed_file, seed_spec
seed_spec: (a list of from 4 to 4 items which are a float)
        seed specification in mm and radius (x y z r)
        flag: -seed %s, position: 2
        mutually_exclusive: seed_file, seed_spec
step_size: (a float)
        Set the step size of the algorithm in mm (default is 0.2).
        flag: -step %s
stop: (a boolean)
        stop track as soon as it enters any of the include regions.
        flag: -stop
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
unidirectional: (a boolean)
        Track from the seed point in one direction only (default is to track
        in both directions).
        flag: -unidirectional

```

Outputs:

```

tracked: (an existing file name)
        output file containing reconstructed tracts

```

StreamlineTrack[Link to code](#)Wraps command **streamtrack**

Performs tractography using one of the following models: 'dt_prob', 'dt_stream', 'sd_prob', 'sd_stream', Where 'dt' stands for diffusion tensor, 'sd' stands for spherical deconvolution, and 'prob' stands for probabilistic.

Example

```

>>> import nipy.interfaces.mrtrix as mrt
>>> strack = mrt.StreamlineTrack()
>>> strack.inputs.inputmodel = 'SD_PROB'
>>> strack.inputs.in_file = 'data.Bfloat'
>>> strack.inputs.seed_file = 'seed_mask.nii'
>>> strack.inputs.mask_file = 'mask.nii'
>>> strack.cmdline
'steamtrack -mask mask.nii -seed seed_mask.nii SD_PROB data.Bfloat data_tracked.
↳tck'
>>> strack.run()

```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    the image containing the source data.The type of data required
    depends on the type of tracking as set in the preceeding argument.
    For DT methods, the base DWI are needed. For SD methods, the SH
    harmonic coefficients of the FOD are needed.
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cutoff_value: (a float)
    Set the FA or FOD amplitude cutoff for terminating tracks (default
    is 0.1).
    flag: -cutoff %s
desired_number_of_tracks: (an integer (int or long))
    Sets the desired number of tracks.The program will continue to
    generate tracks until this number of tracks have been selected and
    written to the output file(default is 100 for *_STREAM methods, 1000
    for *_PROB methods).
    flag: -number %d
do_not_precompute: (a boolean)
    Turns off precomputation of the legendre polynomial values. Warning:
    this will slow down the algorithm by a factor of approximately 4.
    flag: -noprecomputed
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
exclude_file: (an existing file name)
    exclusion file
    flag: -exclude %s
    mutually_exclusive: exclude_file, exclude_spec
exclude_spec: (a list of from 4 to 4 items which are a float)
    exclusion specification in mm and radius (x y z r)
    flag: -exclude %s, position: 2
    mutually_exclusive: exclude_file, exclude_spec
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
include_file: (an existing file name)
    inclusion file
    flag: -include %s
    mutually_exclusive: include_file, include_spec
include_spec: (a list of from 4 to 4 items which are a float)
    inclusion specification in mm and radius (x y z r)
    flag: -include %s, position: 2
    mutually_exclusive: include_file, include_spec
initial_cutoff_value: (a float)
    Sets the minimum FA or FOD amplitude for initiating tracks (default
    is twice the normal cutoff).
    flag: -initcutoff %s
initial_direction: (a list of from 2 to 2 items which are an integer
    (int or long))
    Specify the initial tracking direction as a vector
    flag: -initdirection %s
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipyte default
```

```

    value: DT_STREAM)
    input model type
    flag: %s, position: -3
mask_file: (an existing file name)
    mask file. Only tracks within mask.
    flag: -mask %s
    mutually_exclusive: mask_file, mask_spec
mask_spec: (a list of from 4 to 4 items which are a float)
    Mask specification in mm and radius (x y z r). Tracks will be
    terminated when they leave the ROI.
    flag: -mask %s, position: 2
    mutually_exclusive: mask_file, mask_spec
maximum_number_of_tracks: (an integer (int or long))
    Sets the maximum number of tracks to generate.The program will not
    generate more tracks than this number, even if the desired number of
    tracks hasn't yet been reached(default is 100 x number).
    flag: -maxnum %d
maximum_tract_length: (a float)
    Sets the maximum length of any track in millimeters (default is 200
    mm).
    flag: -length %s
minimum_radius_of_curvature: (a float)
    Set the minimum radius of curvature (default is 2 mm for DT_STREAM,
    0 for SD_STREAM, 1 mm for SD_PROB and DT_PROB)
    flag: -curvature %s
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    flag: -minlength %s
no_mask_interpolation: (a boolean)
    Turns off trilinear interpolation of mask images.
    flag: -nomaskinterp
out_file: (a file name)
    output data file
    flag: %s, position: -1
seed_file: (an existing file name)
    seed file
    flag: -seed %s
    mutually_exclusive: seed_file, seed_spec
seed_spec: (a list of from 4 to 4 items which are a float)
    seed specification in mm and radius (x y z r)
    flag: -seed %s, position: 2
    mutually_exclusive: seed_file, seed_spec
step_size: (a float)
    Set the step size of the algorithm in mm (default is 0.2).
    flag: -step %s
stop: (a boolean)
    stop track as soon as it enters any of the include regions.
    flag: -stop
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
unidirectional: (a boolean)
    Track from the seed point in one direction only (default is to track
    in both directions).
    flag: -unidirectional

```

Outputs:

```
tracked: (an existing file name)
        output file containing reconstructed tracts
```

Tracks2Prob

[Link to code](#)

Wraps command **tracks2prob**

Convert a tract file into a map of the fraction of tracks to enter each voxel - also known as a tract density image (TDI) - in MRtrix's image format (.mif). This can be viewed using MRview or converted to Nifti using MRconvert.

Example

```
>>> import nipyype.interfaces.mrtrix as mrt
>>> tdi = mrt.Tracks2Prob()
>>> tdi.inputs.in_file = 'dwi_CSD_tracked.tck'
>>> tdi.inputs.colour = True
>>> tdi.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        tract file
        flag: %s, position: -2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
colour: (a boolean)
        add colour to the output image according to the direction of the
        tracks.
        flag: -colour, position: 3
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipyype default value: {})
        Environment variables
fraction: (a boolean)
        produce an image of the fraction of fibres through each voxel (as a
        proportion of the total number in the file), rather than the count.
        flag: -fraction, position: 3
ignore_exception: (a boolean, nipyype default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
out_filename: (a file name)
        output data file
        flag: %s, position: -1
output_datatype: ('Bit' or 'Int8' or 'UInt8' or 'Int16' or 'UInt16'
        or 'Int32' or 'UInt32' or 'float32' or 'float64')
        "i.e. Bfloat". Can be "char", "short", "int", "long", "float" or
        "double"
        flag: -datatype %s, position: 2
resample: (a float)
        resample the tracks at regular intervals using Hermite
        interpolation. If omitted, the program will select an appropriate
        interpolation factor automatically.
```



```
    flag: -resample %d, position: 3
template_file: (an existing file name)
    an image file to be used as a template for the output (the output
    image will have the same transform and field of view)
    flag: -template %s, position: 1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
voxel_dims: (a list of from 3 to 3 items which are a float)
    Three comma-separated numbers giving the size of each voxel in mm.
    flag: -vox %s, position: 2
```

Outputs:

```
tract_image: (an existing file name)
    Output tract count or track density image
```


interfaces.mrtrix3.base

MRTrix3Base

[Link to code](#)

Wraps command **None**

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

None

interfaces.mrtrix3.connectivity

BuildConnectome

[Link to code](#)

Wraps command **tck2connectome**

Generate a connectome matrix from a streamlines file and a node parcellation image

Example

```
>>> import nipyre.interfaces.mrtrix3 as mrt
>>> mat = mrt.BuildConnectome()
>>> mat.inputs.in_file = 'tracks.tck'
>>> mat.inputs.in_parcs = 'aparc+aseg.nii'
>>> mat.cmdline
'tck2connectome tracks.tck aparc+aseg.nii connectome.csv'
>>> mat.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input tractography
         flag: %s, position: -3
out_file: (a file name, nipyre default value: connectome.csv)
         output file after processing
         flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipyre default value: {})
         Environment variables
ignore_exception: (a boolean, nipyre default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
in_parcs: (an existing file name)
          parcellation file
          flag: %s, position: -2
in_scalar: (an existing file name)
           provide the associated image for the mean_scalar metric
           flag: -image %s
in_weights: (an existing file name)
            specify a text scalar file containing the streamline weights
            flag: -tck_weights_in %s
keep_unassigned: (a boolean)
                 By default, the program discards the information regarding those
                 streamlines that are not successfully assigned to a node pair. Set
                 this option to keep these values (will be the first row/column in
                 the output matrix)
                 flag: -keep_unassigned
metric: ('count' or 'meanlength' or 'invlength' or 'invnodevolume' or
        'mean_scalar' or 'invlength_invnodevolume')
        specify the edge weight metric
        flag: -metric %s
nthreads: (an integer (int or long))
          number of threads. if zero, the number of available cpus will be
          used
          flag: -nthreads %d
search_forward: (a float)
               project the streamline forwards from the endpoint in search of
```

```

    aparcellation node voxel. Argument is the maximum traversal length
    in mm.
    flag: -assignment_forward_search %f
search_radius: (a float)
    perform a radial search from each streamline endpoint to locate the
    nearest node. Argument is the maximum radius in mm; if no node is
    found within this radius, the streamline endpoint is not assigned to
    any node.
    flag: -assignment_radial_search %f
search_reverse: (a float)
    traverse from each streamline endpoint inwards along the streamline,
    in search of the last node traversed by the streamline. Argument is
    the maximum traversal length in mm (set to 0 to allow search to
    continue to the streamline midpoint).
    flag: -assignment_reverse_search %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
vox_lookup: (a boolean)
    use a simple voxel lookup value at each streamline endpoint
    flag: -assignment_voxel_lookup
zero_diagonal: (a boolean)
    set all diagonal entries in the matrix to zero (these represent
    streamlines that connect to the same node at both ends)
    flag: -zero_diagonal

```

Outputs:

```

out_file: (an existing file name)
    the output response file

```

LabelConfig[Link to code](#)Wraps command **labelconfig**

Re-configure parcellation to be incrementally defined.

Example

```

>>> import nipy.interfaces.mrtrix3 as mrt
>>> labels = mrt.LabelConfig()
>>> labels.inputs.in_file = 'aparc+aseg.nii'
>>> labels.inputs.in_config = 'mrtrix3_labelconfig.txt'
>>> labels.cmdline
'labelconfig aparc+aseg.nii mrtrix3_labelconfig.txt parcellation.mif'
>>> labels.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input anatomical image
    flag: %s, position: -3
out_file: (a file name, nipy default value: parcellation.mif)
    output file after processing
    flag: %s, position: -1

```

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_config: (an existing file name)
    connectome configuration file
    flag: %s, position: -2
lut_aal: (a file name)
    get information from the AAL lookup table (typically
    "ROI_MNI_V4.txt")
    flag: -lut_aal %s
lut_basic: (a file name)
    get information from a basic lookup table consisting of index / name
    pairs
    flag: -lut_basic %s
lut_fs: (a file name)
    get information from a FreeSurfer lookup table (typically
    "FreeSurferColorLUT.txt")
    flag: -lut_freesurfer %s
lut_itksnap: (a file name)
    get information from an ITK - SNAP lookup table (this includes the
    IIT atlas file "LUT_GM.txt")
    flag: -lut_itksnap %s
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    flag: -nthreads %d
spine: (a file name)
    provide a manually-defined segmentation of the base of the spine
    where the streamlines terminate, so that this can become a node in
    the connection matrix.
    flag: -spine %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    the output response file

```

interfaces.mrtrix3.preprocess**ResponseSD**[Link to code](#)Wraps command **dwi2response**

Generate an appropriate response function from the image data for spherical deconvolution.

Example

```

>>> import nipy.interfaces.mrtrix3 as mrt
>>> resp = mrt.ResponseSD()
>>> resp.inputs.in_file = 'dwi.mif'
>>> resp.inputs.in_mask = 'mask.nii.gz'
>>> resp.inputs.grad_fsl = ('bvecs', 'bvals')
>>> resp.cmdline
'dwi2response -fslgrad bvecs bvals -mask mask.nii.gz dwi.mif response.txt'
>>> resp.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         input diffusion weighted images
         flag: %s, position: -2
out_file: (a file name, nipy default value: response.txt)
         output file containing SH coefficients
         flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
bval_scale: ('yes' or 'no')
            specifies whether the b - values should be scaled by the square of
            the corresponding DW gradient norm, as often required for multishell
            or DSI DW acquisition schemes. The default action can also be set in
            the MRtrix config file, under the BValueScaling entry. Valid choices
            are yes / no, true / false, 0 / 1 (default: true).
            flag: -bvalue_scaling %s
disp_mult: (a float)
            dispersion of FOD lobe must not exceed some threshold as determined
            by this multiplier and the FOD dispersion in other single-fibre
            voxels. The threshold is: (mean + (multiplier * (mean - min)));
            default = 1.0. Criterion is only applied in second pass of RF
            estimation.
            flag: -dispersion_multiplier %f
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
grad_file: (an existing file name)
         dw gradient scheme (MRTrix format
         flag: -grad %s
grad_fsl: (a tuple of the form: (an existing file name, an existing
         file name))
         (bvecs, bvals) dw gradient scheme (FSL format
         flag: -fslgrad %s %s
ignore_exception: (a boolean, nipy default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
in_bval: (an existing file name)
         bvals file in FSL format
in_bvec: (an existing file name)
         bvecs file in FSL format
         flag: -fslgrad %s %s
in_mask: (an existing file name)

```

```

        provide initial mask image
        flag: -mask %s
int_mult: (a float)
    integral of FOD lobe must not be outside some range as determined by
    this multiplier and FOD lobe integral in other single-fibre voxels.
    The range is: (mean +- (multiplier * stdev)); default = 2.0.
    Criterion is only applied in second pass of RF estimation.
    flag: -integral_multiplier %f
iterations: (an integer (int or long))
    maximum number of iterations per pass
    flag: -max_iters %d
max_change: (a float)
    maximum percentile change in any response function coefficient; if
    no individual coefficient changes by more than this fraction, the
    algorithm is terminated.
    flag: -max_change %f
max_sh: (an integer (int or long))
    maximum harmonic degree of response function
    flag: -lmax %d
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    flag: -nthreads %d
out_sf: (a file name)
    write a mask containing single-fibre voxels
    flag: -sf %s
shell: (a list of items which are a float)
    specify one or more dw gradient shells
    flag: -shell %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
test_all: (a boolean)
    re-test all voxels at every iteration
    flag: -test_all
vol_ratio: (a float)
    maximal volume ratio between the sum of all other positive lobes in
    the voxel and the largest FOD lobe
    flag: -volume_ratio %f

```

Outputs:

```

out_file: (an existing file name)
    the output response file
out_sf: (a file name)
    mask containing single-fibre voxels

```

interfaces.mrtrix3.reconst**EstimateFOD**[Link to code](#)Wraps command **dwi2fod**

Convert diffusion-weighted images to tensor images

Note that this program makes use of implied symmetries in the diffusion profile. First, the fact the signal attenuation profile is real implies that it has conjugate symmetry, i.e. $Y(l, -m) = Y(l, m)^*$ (where $*$ denotes the

complex conjugate). Second, the diffusion profile should be antipodally symmetric (i.e. $S(x) = S(-x)$), implying that all odd l components should be zero. Therefore, this program only computes the even elements.

Note that the spherical harmonics equations used here differ slightly from those conventionally used, in that the $(-1)^m$ factor has been omitted. This should be taken into account in all subsequent calculations. The spherical harmonic coefficients are stored as follows. First, since the signal attenuation profile is real, it has conjugate symmetry, i.e. $Y(l, -m) = Y(l, m)^*$ (where $*$ denotes the complex conjugate). Second, the diffusion profile should be antipodally symmetric (i.e. $S(x) = S(-x)$), implying that all odd l components should be zero. Therefore, only the even elements are computed.

Note that the spherical harmonics equations used here differ slightly from those conventionally used, in that the $(-1)^m$ factor has been omitted. This should be taken into account in all subsequent calculations. Each volume in the output image corresponds to a different spherical harmonic component. Each volume will correspond to the following:

volume 0: $l = 0, m = 0$ volume 1: $l = 2, m = -2$ (imaginary part of $m=2$ SH) volume 2: $l = 2, m = -1$ (imaginary part of $m=1$ SH) volume 3: $l = 2, m = 0$ volume 4: $l = 2, m = 1$ (real part of $m=1$ SH) volume 5: $l = 2, m = 2$ (real part of $m=2$ SH) etc...

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> fod = mrt.EstimateFOD()
>>> fod.inputs.in_file = 'dwi.mif'
>>> fod.inputs.response = 'response.txt'
>>> fod.inputs.in_mask = 'mask.nii.gz'
>>> fod.inputs.grad_fsl = ('bvecs', 'bvals')
>>> fod.cmdline
'dwi2fod -fslgrad bvecs bvals -mask mask.nii.gz dwi.mif response.txt fods.mif'
>>> fod.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input diffusion weighted images
    flag: %s, position: -3
out_file: (a file name, nipy default value: fods.mif)
    the output spherical harmonics coefficients image
    flag: %s, position: -1
response: (an existing file name)
    a text file containing the diffusion-weighted signal response
    function coefficients for a single fibre population
    flag: %s, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bval_scale: ('yes' or 'no')
    specifies whether the b - values should be scaled by the square of
    the corresponding DW gradient norm, as often required for multishell
    or DSI DW acquisition schemes. The default action can also be set in
    the MRtrix config file, under the BValueScaling entry. Valid choices
    are yes / no, true / false, 0 / 1 (default: true).
    flag: -bvalue_scaling %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
```

```

grad_file: (an existing file name)
    dw gradient scheme (MRTrix format)
    flag: -grad %s
grad_fsl: (a tuple of the form: (an existing file name, an existing
    file name))
    (bvecs, bvals) dw gradient scheme (FSL format)
    flag: -fslgrad %s %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_bval: (an existing file name)
    bvals file in FSL format
in_bvec: (an existing file name)
    bvecs file in FSL format
    flag: -fslgrad %s %s
in_dirs: (an existing file name)
    specify the directions over which to apply the non-negativity
    constraint (by default, the built-in 300 direction set is used).
    These should be supplied as a text file containing the [ az el ]
    pairs for the directions.
    flag: -directions %s
in_mask: (an existing file name)
    provide initial mask image
    flag: -mask %s
max_sh: (an integer (int or long))
    maximum harmonic degree of response function
    flag: -lmax %d
n_iter: (an integer (int or long))
    the maximum number of iterations to perform for each voxel
    flag: -niter %d
neg_lambda: (a float)
    the regularisation parameter lambda that controls the strength of
    the non-negativity constraint
    flag: -neg_lambda %f
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    flag: -nthreads %d
sh_filter: (an existing file name)
    the linear frequency filtering parameters used for the initial
    linear spherical deconvolution step (default = [ 1 1 1 0 0 ]). These
    should be supplied as a text file containing the filtering
    coefficients for each even harmonic order.
    flag: -filter %s
shell: (a list of items which are a float)
    specify one or more dw gradient shells
    flag: -shell %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thres: (a float)
    the threshold below which the amplitude of the FOD is assumed to be
    zero, expressed as an absolute amplitude
    flag: -threshold %f

```

Outputs:

```
out_file: (an existing file name)
          the output response file
```

FitTensor

[Link to code](#)

Wraps command **dwi2tensor**

Convert diffusion-weighted images to tensor images

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> tsr = mrt.FitTensor()
>>> tsr.inputs.in_file = 'dwi.mif'
>>> tsr.inputs.in_mask = 'mask.nii.gz'
>>> tsr.inputs.grad_fsl = ('bvecs', 'bvals')
>>> tsr.cmdline
'dwi2tensor -fslgrad bvecs bvals -mask mask.nii.gz dwi.mif dti.mif'
>>> tsr.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input diffusion weighted images
         flag: %s, position: -2
out_file: (a file name, nipy default value: dti.mif)
         the output diffusion tensor image
         flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
bval_scale: ('yes' or 'no')
            specifies whether the b - values should be scaled by the square of
            the corresponding DW gradient norm, as often required for multishell
            or DSI DW acquisition schemes. The default action can also be set in
            the MRtrix config file, under the BValueScaling entry. Valid choices
            are yes / no, true / false, 0 / 1 (default: true).
            flag: -bvalue_scaling %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
grad_file: (an existing file name)
         dw gradient scheme (MRTrix format)
         flag: -grad %s
grad_fsl: (a tuple of the form: (an existing file name, an existing
         file name))
         (bvecs, bvals) dw gradient scheme (FSL format)
         flag: -fslgrad %s %s
ignore_exception: (a boolean, nipy default value: False)
         Print an error message instead of throwing an exception in case the
         interface fails to run
in_bval: (an existing file name)
         bvals file in FSL format
```

```

in_bvec: (an existing file name)
        bvecs file in FSL format
        flag: -fslgrad %s %s
in_mask: (an existing file name)
        only perform computation within the specified binary brain mask
        image
        flag: -mask %s
method: ('nonlinear' or 'loglinear' or 'sech' or 'rician')
        select method used to perform the fitting
        flag: -method %s
nthreads: (an integer (int or long))
        number of threads. if zero, the number of available cpus will be
        used
        flag: -nthreads %d
reg_term: (a float)
        specify the strength of the regularisation term on the magnitude of
        the tensor elements (default = 5000). This only applies to the non-
        linear methods
        flag: -regularisation %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
        the output DTI file

```

interfaces.mrtrix3.tracking

Tractography

[Link to code](#)Wraps command **tckgen**

Performs streamlines tractography after selecting the appropriate algorithm.

Example

```

>>> import nipype.interfaces.mrtrix3 as mrt
>>> tk = mrt.Tractography()
>>> tk.inputs.in_file = 'fods.mif'
>>> tk.inputs.roi_mask = 'mask.nii.gz'
>>> tk.inputs.seed_sphere = (80, 100, 70, 10)
>>> tk.cmdline
'tckgen -algorithm iFOD2 -mask mask.nii.gz -seed_sphere 80.000000,100.000000,70.
↪000000,10.000000 fods.mif tracked.tck'
>>> tk.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file to be processed
        flag: %s, position: -2
out_file: (a file name, nipype default value: tracked.tck)

```

```

        output file containing tracks
        flag: %s, position: -1

[Optional]
act_file: (an existing file name)
        use the Anatomically-Constrained Tractography framework during
        tracking; provided image must be in the 5TT (five - tissue - type)
        format
        flag: -act %s
algorithm: ('iFOD2' or 'FACT' or 'iFOD1' or 'Nulldist' or 'SD_Stream'
        or 'Tensor_Det' or 'Tensor_Prob', nipy default value: iFOD2)
        tractography algorithm to be used
        flag: -algorithm %s
angle: (a float)
        set the maximum angle between successive steps (default is 90deg x
        stepsize / voxelsize)
        flag: -angle %f
args: (a unicode string)
        Additional parameters to the command
        flag: %s
backtrack: (a boolean)
        allow tracks to be truncated
        flag: -backtrack
bval_scale: ('yes' or 'no')
        specifies whether the b - values should be scaled by the square of
        the corresponding DW gradient norm, as often required for multishell
        or DSI DW acquisition schemes. The default action can also be set in
        the MRtrix config file, under the BValueScaling entry. Valid choices
        are yes / no, true / false, 0 / 1 (default: true).
        flag: -bvalue_scaling %s
crop_at_gmwmi: (a boolean)
        crop streamline endpoints more precisely as they cross the GM-WM
        interface
        flag: -crop_at_gmwmi
cutoff: (a float)
        set the FA or FOD amplitude cutoff for terminating tracks (default
        is 0.1)
        flag: -cutoff %f
cutoff_init: (a float)
        set the minimum FA or FOD amplitude for initiating tracks (default
        is the same as the normal cutoff)
        flag: -initcutoff %f
downsample: (a float)
        downsample the generated streamlines to reduce output file size
        flag: -downsample %f
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
grad_file: (an existing file name)
        dw gradient scheme (MRtrix format)
        flag: -grad %s
grad_fsl: (a tuple of the form: (an existing file name, an existing
        file name))
        (bvecs, bvals) dw gradient scheme (FSL format)
        flag: -fslgrad %s %s
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the

```

```
interface fails to run
in_bval: (an existing file name)
        bvals file in FSL format
in_bvec: (an existing file name)
        bvecs file in FSL format
        flag: -fslgrad %s %s
init_dir: (a tuple of the form: (a float, a float, a float))
        specify an initial direction for the tracking (this should be
        supplied as a vector of 3 comma-separated values
        flag: -initdirection %f,%f,%f
max_length: (a float)
        set the maximum length of any track in mm (default is 100 x
        voxelsize)
        flag: -maxlength %f
max_seed_attempts: (an integer (int or long))
        set the maximum number of times that the tracking algorithm should
        attempt to find an appropriate tracking direction from a given seed
        point
        flag: -max_seed_attempts %d
max_tracks: (an integer (int or long))
        set the maximum number of tracks to generate. The program will not
        generate more tracks than this number, even if the desired number of
        tracks hasn't yet been reached (default is 100 x number)
        flag: -maxnum %d
min_length: (a float)
        set the minimum length of any track in mm (default is 5 x voxelsize)
        flag: -minlength %f
n_samples: (an integer (int or long))
        set the number of FOD samples to take per step for the 2nd order
        (iFOD2) method
        flag: -samples %d
n_tracks: (an integer (int or long))
        set the desired number of tracks. The program will continue to
        generate tracks until this number of tracks have been selected and
        written to the output file
        flag: -number %d
n_trials: (an integer (int or long))
        set the maximum number of sampling trials at each point (only used
        for probabilistic tracking)
        flag: -trials %d
nprecompt: (a boolean)
        do NOT pre-compute legendre polynomial values. Warning: this will
        slow down the algorithm by a factor of approximately 4
        flag: -nprecomputed
nthreads: (an integer (int or long))
        number of threads. if zero, the number of available cpus will be
        used
        flag: -nthreads %d
out_seeds: (a file name)
        output the seed location of all successful streamlines to a file
        flag: -output_seeds %s
power: (an integer (int or long))
        raise the FOD to the power specified (default is 1/nsamples)
        flag: -power %d
roi_excl: (an existing file name or a tuple of the form: (a float, a
        float, a float, a float))
        specify an exclusion region of interest, streamlines that enter ANY
        exclude region will be discarded
```

```

    flag: -exclude %s
roi_incl: (an existing file name or a tuple of the form: (a float, a
    float, a float, a float))
    specify an inclusion region of interest, streamlines must traverse
    ALL inclusion regions to be accepted
    flag: -include %s
roi_mask: (an existing file name or a tuple of the form: (a float, a
    float, a float, a float))
    specify a masking region of interest. If defined, streamlines exiting
    the mask will be truncated
    flag: -mask %s
seed_dynamic: (an existing file name)
    determine seed points dynamically using the SIFT model (must not
    provide any other seeding mechanism). Note that while this seeding
    mechanism improves the distribution of reconstructed streamlines
    density, it should NOT be used as a substitute for the SIFT method
    itself.
    flag: -seed_dynamic %s
seed_gmwmi: (an existing file name)
    seed from the grey matter - white matter interface (only valid if
    using ACT framework)
    flag: -seed_gmwmi %s
    requires: act_file
seed_grid_voxel: (a tuple of the form: (an existing file name, an
    integer (int or long)))
    seed a fixed number of streamlines per voxel in a mask image; place
    seeds on a 3D mesh grid (grid_size argument is per axis; so a
    grid_size of 3 results in 27 seeds per voxel)
    flag: -seed_grid_per_voxel %s %d
    mutually_exclusive: seed_image, seed_rnd_voxel
seed_image: (an existing file name)
    seed streamlines entirely at random within mask
    flag: -seed_image %s
seed_rejection: (an existing file name)
    seed from an image using rejection sampling (higher values = more
    probable to seed from
    flag: -seed_rejection %s
seed_rnd_voxel: (a tuple of the form: (an existing file name, an
    integer (int or long)))
    seed a fixed number of streamlines per voxel in a mask image; random
    placement of seeds in each voxel
    flag: -seed_random_per_voxel %s %d
    mutually_exclusive: seed_image, seed_grid_voxel
seed_sphere: (a tuple of the form: (a float, a float, a float, a
    float))
    spherical seed
    flag: -seed_sphere %f,%f,%f,%f
sph_trait: (a tuple of the form: (a float, a float, a float, a
    float))
    flag: %f,%f,%f,%f
step_size: (a float)
    set the step size of the algorithm in mm (default is 0.1 x
    voxelsize; for iFOD2: 0.5 x voxelsize)
    flag: -step %f
stop: (a boolean)
    stop propagating a streamline once it has traversed all include
    regions
    flag: -stop

```

```
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
unidirectional: (a boolean)
    track from the seed point in one direction only (default is to track
    in both directions)
    flag: -unidirectional
use_rk4: (a boolean)
    use 4th-order Runge-Kutta integration (slower, but eliminates
    curvature overshoot in 1st-order deterministic methods)
    flag: -rk4
```

Outputs:

```
out_file: (an existing file name)
    the output filtered tracks
out_seeds: (a file name)
    output the seed location of all successful streamlines to a file
```

interfaces.mrtrix3.utils

BrainMask

[Link to code](#)

Wraps command **dwi2mask**

Convert a mesh surface to a partial volume estimation image

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> bmsk = mrt.BrainMask()
>>> bmsk.inputs.in_file = 'dwi.mif'
>>> bmsk.cmdline
'dwi2mask dwi.mif brainmask.mif'
>>> bmsk.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input diffusion weighted images
    flag: %s, position: -2
out_file: (a file name, nipy default value: brainmask.mif)
    output brain mask
    flag: %s, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bval_scale: ('yes' or 'no')
    specifies whether the b - values should be scaled by the square of
    the corresponding DW gradient norm, as often required for multishell
    or DSI DW acquisition schemes. The default action can also be set in
    the MRtrix config file, under the BValueScaling entry. Valid choices
```



```

    are yes / no, true / false, 0 / 1 (default: true).
    flag: -bvalue_scaling %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
grad_file: (an existing file name)
    dw gradient scheme (MRTrix format
    flag: -grad %s
grad_fsl: (a tuple of the form: (an existing file name, an existing
    file name))
    (bvecs, bvals) dw gradient scheme (FSL format
    flag: -fslgrad %s %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_bval: (an existing file name)
    bvals file in FSL format
in_bvec: (an existing file name)
    bvecs file in FSL format
    flag: -fslgrad %s %s
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    flag: -nthreads %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (an existing file name)
    the output response file

```

ComputeTDI[Link to code](#)**Wraps command `tckmap`**

Use track data as a form of contrast for producing a high-resolution image.

References

- For TDI or DEC TDI: Calamante, F.; Tournier, J.-D.; Jackson, G. D. & Connelly, A. Track-density imaging (TDI): Super-resolution white matter imaging using whole-brain track-density mapping. *NeuroImage*, 2010, 53, 1233-1243
- If using -contrast length and -stat_vox mean: Pannek, K.; Mathias, J. L.; Bigler, E. D.; Brown, G.; Taylor, J. D. & Rose, S. E. The average pathlength map: A diffusion MRI tractography-derived index for studying brain pathology. *NeuroImage*, 2011, 55, 133-141
- If using -dixel option with TDI contrast only: Smith, R.E., Tournier, J.-D., Calamante, F., Connelly, A. A novel paradigm for automated segmentation of very large whole-brain probabilistic tractography data sets. In *proc. ISMRM*, 2011, 19, 673
- If using -dixel option with any other contrast: Pannek, K., Raffelt, D., Salvado, O., Rose, S. Incorporating directional information in diffusion tractography derived maps: angular track imaging (ATI). In *Proc. ISMRM*, 2012, 20, 1912
- If using -tod option: Dhollander, T., Emsell, L., Van Hecke, W., Maes, F., Sunaert, S., Suetens, P. Track Orientation Density Imaging (TODI) and Track Orientation Distribution (TOD) based tractography. *NeuroImage*,

2014, 94, 312-336

- If using other contrasts / statistics: Calamante, F.; Tournier, J.-D.; Smith, R. E. & Connelly, A. A generalised framework for super-resolution track-weighted imaging. *NeuroImage*, 2012, 59, 2494-2503
 - If using -precise mapping option: Smith, R. E.; Tournier, J.-D.; Calamante, F. & Connelly, A. SIFT: Spherical-deconvolution informed filtering of tractograms. *NeuroImage*, 2013, 67, 298-312 (Appendix 3)
-

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> tdi = mrt.ComputeTDI()
>>> tdi.inputs.in_file = 'dti.mif'
>>> tdi.cmdline
'tckmap dti.mif tdi.mif'
>>> tdi.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input tractography
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
contrast: ('tdi' or 'length' or 'invlength' or 'scalar_map' or
          'scalar_map_conut' or 'fod_amp' or 'curvature')
          define the desired form of contrast for the output image
          flag: -contrast %s
data_type: ('float' or 'unsigned int')
            specify output image data type
            flag: -datatype %s
dixel: (a file name)
        map streamlines todixels within each voxel. Directions are stored
        asazimuth elevation pairs.
        flag: -dixel %s
ends_only: (a boolean)
            only map the streamline endpoints to the image
            flag: -ends_only
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
fwhm_tck: (a float)
          define the statistic for choosing the contribution to be made by
          each streamline as a function of the samples taken along their
          lengths
          flag: -fwhm_tck %f
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
in_map: (an existing file name)
        provide thescalar image map for generating images with 'scalar_map'
        contrasts, or the SHs image for fod_amp
        flag: -image %s
map_zero: (a boolean)
```

```

        if a streamline has zero contribution based on the contrast &
        statistic, typically it is not mapped; use this option to still
        contribute to the map even if this is the case (these non-
        contributing voxels can then influence the mean value in each voxel
        of the map)
        flag: -map_zero
max_tod: (an integer (int or long))
        generate a Track Orientation Distribution (TOD) in each voxel.
        flag: -tod %d
nthreads: (an integer (int or long))
        number of threads. if zero, the number of available cpus will be
        used
        flag: -nthreads %d
out_file: (a file name, nipy default value: tdi.mif)
        output TDI file
        flag: %s, position: -1
precise: (a boolean)
        use a more precise streamline mapping strategy, that accurately
        quantifies the length through each voxel (these lengths are then
        taken into account during TWI calculation)
        flag: -precise
reference: (an existing file name)
        a referenceimage to be used as template
        flag: -template %s
stat_tck: ('mean' or 'sum' or 'min' or 'max' or 'median' or
        'mean_nonzero' or 'gaussian' or 'ends_min' or 'ends_mean' or
        'ends_max' or 'ends_prod')
        define the statistic for choosing the contribution to be made by
        each streamline as a function of the samples taken along their
        lengths.
        flag: -stat_tck %s
stat_vox: ('sum' or 'min' or 'mean' or 'max')
        define the statistic for choosing the finalvoxel intensities for a
        given contrast
        flag: -stat_vox %s
tck_weights: (an existing file name)
        specify a text scalar file containing the streamline weights
        flag: -tck_weights_in %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
upsample: (an integer (int or long))
        upsample the tracks by some ratio using Hermite interpolation before
        mapping
        flag: -upsample %d
use_dec: (a boolean)
        perform mapping in DEC space
        flag: -dec
vox_size: (a list of items which are an integer (int or long))
        voxel dimensions
        flag: -vox %s

```

Outputs:

```

out_file: (a file name)
        output TDI file

```

Generate5tt

[Link to code](#)

Wraps command **5ttgen**

Concatenate segmentation results from FSL FAST and FIRST into the 5TT format required for ACT

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> seg = mrt.Generate5tt()
>>> seg.inputs.in_fast = ['tpm_00.nii.gz',
...                       'tpm_01.nii.gz', 'tpm_02.nii.gz']
>>> seg.inputs.in_first = 'first_merged.nii.gz'
>>> seg.cmdline
'5ttgen tpm_00.nii.gz tpm_01.nii.gz tpm_02.nii.gz first_merged.nii.gz act-5tt.mif'
>>> seg.run()
```

Inputs:

```
[Mandatory]
in_fast: (a list of items which are an existing file name)
        list of PVE images from FAST
        flag: %s, position: -3
out_file: (a file name, nipy default value: act-5tt.mif)
        name of output file
        flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
in_first: (an existing file name)
          combined segmentation file from FIRST
          flag: %s, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          segmentation for ACT in 5tt format
```

Mesh2PVE

[Link to code](#)

Wraps command **mesh2pve**

Convert a mesh surface to a partial volume estimation image

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> m2p = mrt.Mesh2PVE()
>>> m2p.inputs.in_file = 'surf1.vtk'
>>> m2p.inputs.reference = 'dwi.mif'
>>> m2p.inputs.in_first = 'T1.nii.gz'
>>> m2p.cmdline
'mesh2pve -first T1.nii.gz surf1.vtk dwi.mif mesh2volume.nii.gz'
>>> m2p.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input mesh
        flag: %s, position: -3
out_file: (a file name, nipy default value: mesh2volume.nii.gz)
        output file containing SH coefficients
        flag: %s, position: -1
reference: (an existing file name)
        input reference image
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
in_first: (an existing file name)
          indicates that the mesh file is provided by FSL FIRST
          flag: -first %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
          the output response file
```

TCK2VTK

[Link to code](#)

Wraps command **tck2vtk**

Convert a track file to a vtk format, cave: coordinates are in XYZ coordinates not reference

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> vtk = mrt.TCK2VTK()
>>> vtk.inputs.in_file = 'tracks.tck'
>>> vtk.inputs.reference = 'b0.nii'
>>> vtk.cmdline
'tck2vtk -image b0.nii tracks.tck tracks.vtk'
>>> vtk.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input tractography
        flag: %s, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
nthreads: (an integer (int or long))
           number of threads. if zero, the number of available cpus will be
           used
           flag: -nthreads %d
out_file: (a file name, nipy default value: tracks.vtk)
          output VTK file
          flag: %s, position: -1
reference: (an existing file name)
           if specified, the properties of this image will be used to convert
           track point positions from real (scanner) coordinates into image
           coordinates (in mm).
           flag: -image %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
voxel: (an existing file name)
       if specified, the properties of this image will be used to convert
       track point positions from real (scanner) coordinates into image
       coordinates.
       flag: -image %s
```

Outputs:

```
out_file: (a file name)
          output VTK file
```

TensorMetrics

[Link to code](#)

Wraps command **tensor2metric**

Compute metrics from tensors

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> comp = mrt.TensorMetrics()
>>> comp.inputs.in_file = 'dti.mif'
>>> comp.inputs.out_fa = 'fa.mif'
>>> comp.cmdline
'tensor2metric -fa fa.mif dti.mif'
>>> comp.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input DTI image
        flag: %s, position: -1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
component: (a list of items which are any value)
            specify the desired eigenvalue/eigenvector(s). Note that several
            eigenvalues can be specified as a number sequence
            flag: -num %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
in_mask: (an existing file name)
          only perform computation within the specified binary brain mask
          image
          flag: -mask %s
modulate: ('FA' or 'none' or 'eval')
           how to modulate the magnitude of the eigenvectors
           flag: -modulate %s
out_adc: (a file name)
          output ADC file
          flag: -adc %s
out_eval: (a file name)
           output selected eigenvalue(s) file
           flag: -value %s
out_evec: (a file name)
           output selected eigenvector(s) file
           flag: -vector %s
out_fa: (a file name)
         output FA file
         flag: -fa %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_adc: (a file name)
         output ADC file
out_eval: (a file name)
         output selected eigenvalue(s) file
out_evec: (a file name)
         output selected eigenvector(s) file
out_fa: (a file name)
        output FA file
```


interfaces.niftyfit.asl

FitAsl

[Link to code](#)

Wraps command **fit_asl**

Interface for executable **fit_asl** from Niftyfit platform.

Use NiftyFit to perform ASL fitting.

ASL fitting routines (following EU Cost Action White Paper recommendations) Fits Cerebral Blood Flow maps in the first instance.

[Source code](#)

Examples

```
>>> from nipy.interfaces import niftyfit
>>> node = niftyfit.FitAsl()
>>> node.inputs.source_file = 'asl.nii.gz'
>>> node.cmdline
'fit_asl -source asl.nii.gz -cbf asl_cbf.nii.gz -error asl_error.nii.gz -syn asl_
↪syn.nii.gz'
```

Inputs:

```
[Mandatory]
source_file: (a file name)
    Filename of the 4D ASL (control/label) source image (mandatory).
    flag: -source %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cbf_file: (a file name)
    Filename of the Cerebral Blood Flow map (in ml/100g/min).
    flag: -cbf %s
dpld: (a float)
```

```

        Difference in labelling delay per slice [0.0 ms/slice].
        flag: -dPLD %f
dt_inv2: (a float)
        Difference in inversion time per slice [0ms/slice].
        flag: -dTinv2 %f
eff: (a float)
        Labelling efficiency [0.99 (pasl), 0.85 (pcasl)], ensure any
        background suppression pulses are included in -eff
        flag: -eff %f
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
error_file: (a file name)
        Filename of the CBF error map.
        flag: -error %s
gm_plasma: (a float)
        Plasma/GM water partition [0.95ml/g].
        flag: -gmL %f
gm_t1: (a float)
        T1 of GM [1150ms].
        flag: -gmT1 %f
gm_ttt: (a float)
        Time to GM [ATT+0ms].
        flag: -gmTTT %f
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
ir_output: (a file name)
        Output of [1,2,5]s Inversion Recovery fitting.
        flag: -IRoutput %s
ir_volume: (a file name)
        Filename of a [1,2,5]s Inversion Recovery volume (T1/M0 fitting
        carried out internally).
        flag: -IRvolume %s
ldd: (a float)
        Labelling Duration [1800ms].
        flag: -LDD %f
m0map: (a file name)
        Filename of the estimated input M0 map.
        flag: -m0map %s
m0mape: (a file name)
        Filename of the estimated input M0 map error.
        flag: -m0mape %s
mask: (a file name)
        Filename of image mask.
        flag: -mask %s, position: 2
mul: (a float)
        Multiply CBF by this value (e.g. if CL are mislabelled use -1.0).
        flag: -mul %f
mulgm: (a boolean)
        Multiply CBF by segmentation [Off].
        flag: -sig
out: (a float)
        Outlier rejection for multi CL volumes (enter z-score threshold
        (e.g. 2.5)) [off].
        flag: -out %f
pasl: (a boolean)

```

```

    Fit PASL ASL data [default]
    flag: -pasl
pcasl: (a boolean)
    Fit PCASL ASL data
    flag: -pcasl
plasma_coeff: (a float)
    Single plasma/tissue partition coefficient [0.9ml/g].
    flag: -L %f
pld: (a float)
    Post Labelling Delay [2000ms].
    flag: -PLD %f
pv0: (an integer (int or long))
    Simple PV correction ( $CBF = v_g * CBF_g + v_w * CBF_w$ , with  $CBF_w = f * CBF_g$ )
    [0.25].
    flag: -pv0 %d
pv2: (an integer (int or long))
    In plane PV kernel size [3x3].
    flag: -pv2 %d
pv3: (a tuple of the form: (an integer (int or long), an integer (int
    or long), an integer (int or long)))
    3D kernel size [3x3x1].
    flag: -pv3 %d %d %d
pv_threshold: (a boolean)
    Set PV threshold for switching off LSQR [0.05].
    flag: -pvthreshold
seg: (a file name)
    Filename of the 4D segmentation (in ASL space) for L/T1 estimation
    and PV correction {WM,GM,CSF}.
    flag: -seg %s
segstyle: (a boolean)
    Set CBF as [gm,wm] not [wm,gm].
    flag: -segstyle
sig: (a boolean)
    Use sigmoid to estimate L from T1:  $L(T1|gmL,wmL)$  [Off].
    flag: -sig
syn_file: (a file name)
    Filename of the synthetic ASL data.
    flag: -syn %s
t1_art_cmp: (a float)
    T1 of arterial component [1650ms].
    flag: -T1a %f
t1map: (a file name)
    Filename of the estimated input T1 map (in ms).
    flag: -t1map %s
t_inv1: (a float)
    Saturation pulse time [800ms].
    flag: -Tinv1 %f
t_inv2: (a float)
    Inversion time [2000ms].
    flag: -Tinv2 %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
wm_plasma: (a float)
    Plasma/WM water partition [0.82ml/g].
    flag: -wmL %f
wm_t1: (a float)

```

```
T1 of WM [800ms].
flag: -wmT1 %f
wm_ttt: (a float)
Time to WM [ATT+0ms].
flag: -wmTTT %f
```

Outputs:

```
cbf_file: (a file name)
    Filename of the Cerebral Blood Flow map (in ml/100g/min) .
error_file: (a file name)
    Filename of the CBF error map.
syn_file: (a file name)
    Filename of the synthetic ASL data.
```

interfaces.niftyfit.base

NiftyFitCommand

[Link to code](#)

Wraps command **None**

Base support interface for NiftyFit commands.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

None

interfaces.niftyfit.dwi

DwiTool

[Link to code](#)

Wraps command **dwi_tool**

Interface for executable dwi_tool from Niftyfit platform.

Use DwiTool.

Diffusion-Weighted MR Prediction. Predicts DWI from previously fitted models and calculates model derived maps.

Source code

Examples

```
>>> from nipy.interfaces import niftyfit
>>> dwi_tool = niftyfit.DwiTool(dti_flag=True)
>>> dwi_tool.inputs.source_file = 'dwi.nii.gz'
>>> dwi_tool.inputs.bvec_file = 'bvecs'
>>> dwi_tool.inputs.bval_file = 'bvals'
>>> dwi_tool.inputs.mask_file = 'mask.nii.gz'
>>> dwi_tool.inputs.b0_file = 'b0.nii.gz'
>>> dwi_tool.inputs.rgbmap_file = 'rgb_map.nii.gz'
>>> dwi_tool.cmdline
'dwi_tool -source dwi.nii.gz -bval bvals -bvec bvecs -b0 b0.nii.gz -mask mask.nii.
↪gz -dti -famap dwi_famap.nii.gz -logdti2 dwi_logdti2.nii.gz -mcmmap dwi_mcmmap.
↪nii.gz -mdmap dwi_mdmap.nii.gz -rgbmap rgb_map.nii.gz -syn dwi_syn.nii.gz -
↪v1map dwi_v1map.nii.gz'
```

Inputs:

```
[Mandatory]
bval_file: (a file name)
    The file containing the bvalues of the source DWI.
    flag: -bval %s, position: 2
source_file: (a file name)
    The source image containing the fitted model.
    flag: -source %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
b0_file: (a file name)
    The B0 image corresponding to the source DWI
    flag: -b0 %s, position: 4
ball_flag: (a boolean)
    Input is a ball and stick model.
    flag: -ball, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
    ballv_flag, nod_flag, nodv_flag
ballv_flag: (a boolean)
    Input is a ball and stick model with optimised PDD.
    flag: -ballv, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
    ball_flag, nod_flag, nodv_flag
bvec_file: (a file name)
    The file containing the bvectors of the source DWI.
    flag: -bvec %s, position: 3
diso_val: (a float)
    Isotropic diffusivity for -nod [3e-3]
    flag: -diso %f
dpr_val: (a float)
    Parallel diffusivity for -nod [1.7e-3].
    flag: -dpr %f
dti_flag: (a boolean)
    Input is a tensor model diag/off-diag.
    flag: -dti, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag2, ball_flag,
```

```

    ballv_flag, nod_flag, nodv_flag
dti_flag2: (a boolean)
    Input is a tensor model lower triangular
    flag: -dti2, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag,
    ballv_flag, nod_flag, nodv_flag
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
famap_file: (a file name)
    Filename of FA map
    flag: -famap %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
ivim_flag: (a boolean)
    Inputs is an IVIM model to non-directional data.
    flag: -ivim, position: 6
    mutually_exclusive: mono_flag, dti_flag, dti_flag2, ball_flag,
    ballv_flag, nod_flag, nodv_flag
logdti_file: (a file name)
    Filename of output logdti map.
    flag: -logdti2 %s
mask_file: (a file name)
    The image mask
    flag: -mask %s, position: 5
mcmap_file: (a file name)
    Filename of multi-compartment model parameter map (-ivim,-ball,-nod)
    flag: -mcmap %s
mdmap_file: (a file name)
    Filename of MD map/ADC
    flag: -mdmap %s
mono_flag: (a boolean)
    Input is a single exponential to non-directional data [default with
    no b-vectors]
    flag: -mono, position: 6
    mutually_exclusive: ivim_flag, dti_flag, dti_flag2, ball_flag,
    ballv_flag, nod_flag, nodv_flag
nod_flag: (a boolean)
    Input is a NODDI model
    flag: -nod, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
    ball_flag, ballv_flag, nodv_flag
nodv_flag: (a boolean)
    Input is a NODDI model with optimised PDD
    flag: -nodv, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
    ball_flag, ballv_flag, nod_flag
rgbmap_file: (a file name)
    Filename of colour FA map.
    flag: -rgbmap %s
syn_file: (a file name)
    Filename of synthetic image. Requires: bvec_file/b0_file.
    flag: -syn %s
    requires: bvec_file, b0_file
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately

```

```

        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
v1map_file: (a file name)
    Filename of PDD map [x,y,z]
flag: -v1map %s

```

Outputs:

```

f1map_file: (a file name)
    Filename of FA map
logdti_file: (a file name)
    Filename of output logdti map
mcmap_file: (a file name)
    Filename of multi-compartment model parameter map (-ivim,-ball,-nod)
mdmap_file: (a file name)
    Filename of MD map/ADC
rgbmap_file: (a file name)
    Filename of colour FA map
syn_file: (a file name)
    Filename of synthetic image
v1map_file: (a file name)
    Filename of PDD map [x,y,z]

```

FitDwi[Link to code](#)Wraps command **fit_dwi**

Interface for executable fit_dwi from Niftyfit platform.

Use NiftyFit to perform diffusion model fitting.

Diffusion-weighted MR Fitting. Fits DWI parameter maps to multi-shell, multi-directional data.

[Source code](#)**Examples**

```

>>> from nipy.interfaces import niftyfit
>>> fit_dwi = niftyfit.FitDwi(dti_flag=True)
>>> fit_dwi.inputs.source_file = 'dwi.nii.gz'
>>> fit_dwi.inputs.bvec_file = 'bvecs'
>>> fit_dwi.inputs.bval_file = 'bvals'
>>> fit_dwi.inputs.rgbmap_file = 'rgb.nii.gz'
>>> fit_dwi.cmdline
'fit_dwi -source dwi.nii.gz -bval bvals -bvec bvecs -dti -error dwi_error.nii.gz -
↳f1map dwi_f1map.nii.gz -mcmap dwi_mcmap.nii.gz -mcout dwi_mcout.txt -mdmap dwi_
↳mdmap.nii.gz -nodiff dwi_no_diff.nii.gz -res dwi_resmap.nii.gz -rgbmap rgb.nii.
↳gz -syn dwi_syn.nii.gz -tenmap2 dwi_tenmap2.nii.gz -v1map dwi_v1map.nii.gz'

```

Inputs:

```

[Mandatory]
bval_file: (a file name)
    The file containing the bvalues of the source DWI.
    flag: -bval %s, position: 2
bvec_file: (a file name)
    The file containing the bvecs of the source DWI.
    flag: -bvec %s, position: 3
source_file: (a file name)
    The source image containing the dwi data.

```

```
    flag: -source %s, position: 1

[Optional]
acceptance: (a float)
    Fraction of iterations to accept [0.23].
    flag: -accpetance %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
ball_flag: (a boolean)
    Fit the ball and stick model.
    flag: -ball, position: 4
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, ballv_flag,
        nod_flag, nodv_flag
ballv_flag: (a boolean)
    Fit the ball and stick model with optimised PDD.
    flag: -ballv, position: 4
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag,
        nod_flag, nodv_flag
cov_file: (a file name)
    Filename of ithe nc*nc covariance matrix [I]
    flag: -cov %s
csf_t2_val: (a float)
    CSF T2 value [400ms].
    flag: -csfT2 %f
diso_val: (a float)
    Isotropic diffusivity for -nod [3e-3]
    flag: -diso %f
dpr_val: (a float)
    Parallel diffusivity for -nod [1.7e-3].
    flag: -dpr %f
dti_flag: (a boolean)
    Fit the tensor model [default with b-vectors].
    flag: -dti, position: 4
    mutually_exclusive: mono_flag, ivim_flag, ball_flag, ballv_flag,
        nod_flag, nodv_flag
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
error_file: (a file name)
    Filename of parameter error maps.
    flag: -error %s
famap_file: (a file name)
    Filename of FA map
    flag: -famap %s
gn_flag: (a boolean)
    Use Gauss-Newton algorithm [Levenberg-Marquardt].
    flag: -gn
    mutually_exclusive: wls_flag
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
ivim_flag: (a boolean)
    Fit IVIM model to non-directional data.
    flag: -ivim, position: 4
    mutually_exclusive: mono_flag, dti_flag, ball_flag, ballv_flag,
        nod_flag, nodv_flag
```



```

lm_vals: (a tuple of the form: (a float, a float))
    LM parameters (initial value, decrease rate) [100,1.2].
    flag: -lm %f %f
    requires: gn_flag
mask_file: (a file name)
    The image mask
    flag: -mask %s
maxit_val: (an integer (int or long))
    Maximum number of non-linear LSQR iterations [100x2 passes]
    flag: -maxit %d
    requires: gn_flag
mcmmap_file: (a file name)
    Filename of multi-compartment model parameter map (-ivim,-ball,-nod)
    flag: -mcmmap %s
    requires: nodv_flag
mcmaxit: (an integer (int or long))
    Number of iterations to run [10,000].
    flag: -mcmaxit %d
mcout: (a file name)
    Filename of mc samples (ascii text file)
    flag: -mcout %s
mcsamples: (an integer (int or long))
    Number of samples to keep [100].
    flag: -mcsamples %d
mdmap_file: (a file name)
    Filename of MD map/ADC
    flag: -mdmap %s
mono_flag: (a boolean)
    Fit single exponential to non-directional data [default with no
    b-vectors]
    flag: -mono, position: 4
    mutually_exclusive: ivim_flag, dti_flag, ball_flag, ballv_flag,
    nod_flag, nodv_flag
nod_flag: (a boolean)
    Fit the NODDI model
    flag: -nod, position: 4
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag,
    ballv_flag, nodv_flag
nodiff_file: (a file name)
    Filename of average no diffusion image.
    flag: -nodiff %s
nodv_flag: (a boolean)
    Fit the NODDI model with optimised PDD
    flag: -nodv, position: 4
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag,
    ballv_flag, nod_flag
perf_thr: (a float)
    Threshold for perfusion/diffusion effects [100].
    flag: -perfthreshold %f
prior_file: (a file name)
    Filename of parameter priors for -ball and -nod.
    flag: -prior %s
res_file: (a file name)
    Filename of model residual map.
    flag: -res %s
rgbmap_file: (a file name)
    Filename of colour-coded FA map
    flag: -rgbmap %s

```

```
    requires: dti_flag
rot_sform_flag: (an integer (int or long))
    Rotate the output tensors according to the q/s form of the image
    (resulting tensors will be in mm coordinates, default: 0).
    flag: -rotsform %d
slice_no: (an integer (int or long))
    Fit to single slice number.
    flag: -slice %d
swls_val: (a float)
    Use location-weighted least squares for DTI fitting [3x3 Gaussian]
    flag: -swls %f
syn_file: (a file name)
    Filename of synthetic image.
    flag: -syn %s
te_file: (a file name)
    Filename of TEs (ms).
    flag: -TE %s
    mutually_exclusive: te_file
te_value: (a file name)
    Value of TEs (ms).
    flag: -TE %s
    mutually_exclusive: te_file
ten_type: ('lower-tri' or 'diag-off-diag', nipy default value:
    lower-tri)
    Use lower triangular (tenmap2) or diagonal, off-diagonal tensor
    format
tenmap2_file: (a file name)
    Filename of tensor map [lower tri]
    flag: -tenmap2 %s
    requires: dti_flag
tenmap_file: (a file name)
    Filename of tensor map [diag,offdiag].
    flag: -tenmap %s
    requires: dti_flag
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
vlmap_file: (a file name)
    Filename of PDD map [x,y,z]
    flag: -vlmap %s
vb_flag: (a boolean)
    Use Variational Bayes fitting with known prior (currently identity
    covariance...).
    flag: -vb
voxel: (a tuple of the form: (an integer (int or long), an integer
    (int or long), an integer (int or long)))
    Fit to single voxel only.
    flag: -voxel %d %d %d
wls_flag: (a boolean)
    Use Variational Bayes fitting with known prior (currently identity
    covariance...).
    flag: -wls
    mutually_exclusive: gn_flag
wm_t2_val: (a float)
    White matter T2 value [80ms].
    flag: -wmT2 %f
```

Outputs:

```

error_file: (a file name)
    Filename of parameter error maps
fomap_file: (a file name)
    Filename of FA map
mcomap_file: (a file name)
    Filename of multi-compartment model parameter map
    (-ivim, -ball, -nod).
mcout: (a file name)
    Filename of mc samples (ascii text file)
mdmap_file: (a file name)
    Filename of MD map/ADC
nodiff_file: (a file name)
    Filename of average no diffusion image.
res_file: (a file name)
    Filename of model residual map
rgbmap_file: (a file name)
    Filename of colour FA map
syn_file: (a file name)
    Filename of synthetic image
tenmap2_file: (a file name)
    Filename of tensor map [lower tri]
tenmap_file: (a file name)
    Filename of tensor map
vlmap_file: (a file name)
    Filename of PDD map [x,y,z]

```

interfaces.niftyfit.qt1

FitQt1

[Link to code](#)

Wraps command **fit_qt1**

Interface for executable fit_qt1 from Niftyfit platform.

Use NiftyFit to perform Qt1 fitting.

T1 Fitting Routine (To inversion recovery or spgr data). Fits single component T1 maps in the first instance.

[Source code](#)

Examples

```

>>> from nipy.interfaces.niftyfit import FitQt1
>>> fit_qt1 = FitQt1()
>>> fit_qt1.inputs.source_file = 'TI4D.nii.gz'
>>> fit_qt1.cmdline
'fit_qt1 -source TI4D.nii.gz -comp TI4D_comp.nii.gz -error TI4D_error.nii.gz -
↪m0map TI4D_m0map.nii.gz -mcomap TI4D_mcomap.nii.gz -res TI4D_res.nii.gz -syn TI4D_
↪syn.nii.gz -tlmap TI4D_tlmap.nii.gz'

```

Inputs:

```

[Mandatory]
source_file: (an existing file name)
    Filename of the 4D Multi-Echo T1 source image.
    flag: -source %s, position: 1

[Optional]
acceptance: (a float)

```

```

        Fraction of iterations to accept [0.23].
        flag: -acceptance %f
args: (a unicode string)
        Additional parameters to the command
        flag: %s
blmap: (a file name)
        Filename of B1 estimate for fitting (or include in prior).
        flag: -blmap %s
comp_file: (a file name)
        Filename of the estimated multi-component T1 map.
        flag: -comp %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
error_file: (a file name)
        Filename of the error map (symmetric matrix, [Diag,OffDiag]).
        flag: -error %s
flips: (a list of items which are a float)
        Flip angles
        flag: -flips %s
flips_list: (a file name)
        Filename of list of pre-defined flip angles (deg).
        flag: -fliplist %s
gn_flag: (a boolean)
        Use Gauss-Newton algorithm [Levenberg-Marquardt].
        flag: -gn, position: 8
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
ir_flag: (a boolean)
        Inversion Recovery fitting [default].
        flag: -IR, position: 13
lm_val: (a tuple of the form: (a float, a float))
        Set LM parameters (initial value, decrease rate) [100,1.2].
        flag: -lm %f %f, position: 7
m0map_file: (a file name)
        Filename of the estimated input M0 map.
        flag: -m0map %s
mask: (an existing file name)
        Filename of image mask.
        flag: -mask %s, position: 2
maxit: (an integer (int or long))
        NLSQR iterations [100].
        flag: -maxit %d, position: 11
mcmap_file: (a file name)
        Filename of the estimated output multi-parameter map.
        flag: -mcmap %s
mcmaxit: (an integer (int or long))
        Number of iterations to run [10,000].
        flag: -mcmaxit %d
mcout: (a file name)
        Filename of mc samples (ascii text file)
        flag: -mcout %s
mcsamples: (an integer (int or long))
        Number of samples to keep [100].
        flag: -mcsamples %d
nb_comp: (an integer (int or long))

```

```

        Number of components to fit [1] (currently IR/SR only)
        flag: -nc %d, position: 6
prior: (an existing file name)
        Filename of parameter prior.
        flag: -prior %s, position: 3
res_file: (a file name)
        Filename of the model fit residuals
        flag: -res %s
slice_no: (an integer (int or long))
        Fit to single slice number.
        flag: -slice %d, position: 9
spgr: (a boolean)
        Spoiled Gradient Echo fitting
        flag: -SPGR
sr_flag: (a boolean)
        Saturation Recovery fitting [default].
        flag: -SR, position: 12
syn_file: (a file name)
        Filename of the synthetic ASL data.
        flag: -syn %s
t1_list: (a file name)
        Filename of list of pre-defined T1s
        flag: -T1list %s
t1map_file: (a file name)
        Filename of the estimated output T1 map (in ms).
        flag: -t1map %s
t1max: (a float)
        Maximum tissue T1 value [4000ms].
        flag: -T1max %f
t1min: (a float)
        Minimum tissue T1 value [400ms].
        flag: -T1min %f
te_value: (a float)
        TE Echo Time [0ms!].
        flag: -TE %f, position: 4
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
tis: (a list of items which are a float)
        Inversion times for T1 data [1s,2s,5s].
        flag: -TIs %s, position: 14
tis_list: (a file name)
        Filename of list of pre-defined TIs.
        flag: -T1list %s
tr_value: (a float)
        TR Repetition Time [10s!].
        flag: -TR %f, position: 5
voxel: (a tuple of the form: (an integer (int or long), an integer
        (int or long), an integer (int or long)))
        Fit to single voxel only.
        flag: -voxel %d %d %d, position: 10

```

Outputs:

```

comp_file: (a file name)
        Filename of the estimated multi-component T1 map.
error_file: (a file name)

```

```
        Filename of the error map (symmetric matrix, [Diag,OffDiag])
m0map_file: (a file name)
        Filename of the m0 map
mcmap_file: (a file name)
        Filename of the estimated output multi-parameter map
res_file: (a file name)
        Filename of the model fit residuals
syn_file: (a file name)
        Filename of the synthetic ASL data
t1map_file: (a file name)
        Filename of the estimated output T1 map (in ms)
```

interfaces.niftyreg.base

NiftyRegCommand

[Link to code](#)

Wraps command **None**

Base support interface for NiftyReg commands.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
omp_core_val: (an integer (int or long), nipy default value: 1)
    Number of openmp thread to use
    flag: -omp %i
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

None

get_custom_path()

[Link to code](#)

interfaces.niftyreg.reg

RegAladin

[Link to code](#)

Wraps command **reg_aladin**

Interface for executable reg_aladin from NiftyReg platform.

Block Matching algorithm for symmetric global registration. Based on Modat et al., “Global image registration using asymmetric block-matching approach” J. Med. Img. 1(2) 024003, 2014, doi: 10.1117/1.JMI.1.2.024003

[Source code](#)

Examples

```

>>> from nipy.interfaces import niftyreg
>>> node = niftyreg.RegAladin()
>>> node.inputs.ref_file = 'im1.nii'
>>> node.inputs.flo_file = 'im2.nii'
>>> node.inputs.rmask_file = 'mask.nii'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_aladin -aff im2_aff.txt -flo im2.nii -omp 4 -ref im1.nii -res im2_res.nii.gz -rmask mask.nii'

```

Inputs:

```

[Mandatory]
flo_file: (an existing file name)
    The input floating/source image
    flag: -flo %s
ref_file: (an existing file name)
    The input reference/target image
    flag: -ref %s

[Optional]
aff_direct_flag: (a boolean)
    Directly optimise the affine parameters
    flag: -affDirect
aff_file: (a file name)
    The output affine matrix file
    flag: -aff %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cog_flag: (a boolean)
    Use the masks centre of mass to initialise the transformation
    flag: -cog
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
flo_low_val: (a float)
    Lower threshold value on floating image
    flag: -floLowThr %f
flo_up_val: (a float)
    Upper threshold value on floating image
    flag: -floUpThr %f
fmask_file: (an existing file name)
    The input floating mask

```



```

        flag: -fmask %s
gpuid_val: (an integer (int or long))
    Device to use id
    flag: -gpuid %i
i_val: (a long integer >= 0)
    Percent of inlier blocks
    flag: -pi %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_aff_file: (an existing file name)
    The input affine transformation
    flag: -inaff %s
ln_val: (a long integer >= 0)
    Number of resolution levels to create
    flag: -ln %d
lp_val: (a long integer >= 0)
    Number of resolution levels to perform
    flag: -lp %d
maxit_val: (a long integer >= 0)
    Maximum number of iterations
    flag: -maxit %d
nac_flag: (a boolean)
    Use nifti header to initialise transformation
    flag: -nac
nosym_flag: (a boolean)
    Turn off symmetric registration
    flag: -noSym
omp_core_val: (an integer (int or long), nipy default value: 1)
    Number of openmp thread to use
    flag: -omp %i
platform_val: (an integer (int or long))
    Platform index
    flag: -platf %i
ref_low_val: (a float)
    Lower threshold value on reference image
    flag: -refLowThr %f
ref_up_val: (a float)
    Upper threshold value on reference image
    flag: -refUpThr %f
res_file: (a file name)
    The affine transformed floating image
    flag: -res %s
rig_only_flag: (a boolean)
    Do only a rigid registration
    flag: -rigOnly
rmask_file: (an existing file name)
    The input reference mask
    flag: -rmask %s
smoo_f_val: (a float)
    Amount of smoothing to apply to floating image
    flag: -smooF %f
smoo_r_val: (a float)
    Amount of smoothing to apply to reference image
    flag: -smooR %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display

```

```

        output, `file` - writes output to file, `none` - output is ignored
v_val: (a long integer >= 0)
        Percent of blocks that are active
        flag: -pv %d
verbosity_off_flag: (a boolean)
        Turn off verbose output
        flag: -voff

```

Outputs:

```

aff_file: (a file name)
        The output affine file
avg_output: (a string)
        Output string in the format for reg_average
res_file: (a file name)
        The output transformed image

```

RegF3D[Link to code](#)Wraps command **reg_f3d**

Interface for executable reg_f3d from NiftyReg platform.

Fast Free-Form Deformation (F3D) algorithm for non-rigid registration. Initially based on Modat et al., “Fast Free-Form Deformation using graphics processing units”, CMPB, 2010

[Source code](#)**Examples**

```

>>> from nipy.interfaces import niftyreg
>>> node = niftyreg.RegF3D()
>>> node.inputs.ref_file = 'im1.nii'
>>> node.inputs.flo_file = 'im2.nii'
>>> node.inputs.rmask_file = 'mask.nii'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_f3d -cpp im2_cpp.nii.gz -flo im2.nii -omp 4 -ref im1.nii -res im2_res.nii.gz_
↪-rmask mask.nii'

```

Inputs:

```

[Mandatory]
flo_file: (an existing file name)
        The input floating/source image
        flag: -flo %s
ref_file: (an existing file name)
        The input reference/target image
        flag: -ref %s

[Optional]
aff_file: (an existing file name)
        The input affine transformation file
        flag: -aff %s
amc_flag: (a boolean)
        Use additive NMI
        flag: -amc
args: (a unicode string)
        Additional parameters to the command

```

```

        flag: %s
be_val: (a float)
        Bending energy value
        flag: -be %f
cpp_file: (a file name)
        The output CPP file
        flag: -cpp %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
fbn2_val: (a tuple of the form: (a long integer >= 0, a long integer
        >= 0))
        Number of bins in the histogram for reference image for given time
        point
        flag: -fbn %d %d
fbn_val: (a long integer >= 0)
        Number of bins in the histogram for reference image
        flag: --fbn %d
flo_smooth_val: (a float)
        Smoothing kernel width for floating image
        flag: -smooF %f
flwth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
        Lower threshold for floating image at the specified time point
        flag: -fLwTh %d %f
flwth_thr_val: (a float)
        Lower threshold for floating image
        flag: --fLwTh %f
fmask_file: (an existing file name)
        Floating image mask
        flag: -fmask %s
fupth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
        Upper threshold for floating image at the specified time point
        flag: -fUpTh %d %f
fupth_thr_val: (a float)
        Upper threshold for floating image
        flag: --fUpTh %f
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
incpp_file: (an existing file name)
        The input cpp transformation file
        flag: -incpp %s
jl_val: (a float)
        Log of jacobian of deformation penalty value
        flag: -jl %f
kld2_flag: (a long integer >= 0)
        Use KL divergence as the similarity measure for a given time point
        flag: -kld %d
kld_flag: (a boolean)
        Use KL divergence as the similarity measure
        flag: --kld
le_val: (a float)
        Linear elasticity penalty term
        flag: -le %f
ln_val: (a long integer >= 0)
        Number of resolution levels to create
        flag: -ln %d

```

```
lncc2_val: (a tuple of the form: (a long integer >= 0, a float))
    SD of the Gaussian for computing LNCC for a given time point
    flag: -lncc %d %f
lncc_val: (a float)
    SD of the Gaussian for computing LNCC
    flag: --lncc %f
lp_val: (a long integer >= 0)
    Number of resolution levels to perform
    flag: -lp %d
maxit_val: (a long integer >= 0)
    Maximum number of iterations per level
    flag: -maxit %d
nmi_flag: (a boolean)
    use NMI even when other options are specified
    flag: --nmi
no_app_jl_flag: (a boolean)
    Do not approximate the log of jacobian penalty at control points
    only
    flag: -noAppJL
noconj_flag: (a boolean)
    Use simple GD optimization
    flag: -noConj
nopy_flag: (a boolean)
    Do not use the multiresolution approach
    flag: -nopy
nox_flag: (a boolean)
    Don't optimise in x direction
    flag: -nox
noy_flag: (a boolean)
    Don't optimise in y direction
    flag: -noy
noz_flag: (a boolean)
    Don't optimise in z direction
    flag: -noz
omp_core_val: (an integer (int or long), nipyre default value: 1)
    Number of openmp thread to use
    flag: -omp %i
pad_val: (a float)
    Padding value
    flag: -pad %f
pert_val: (a long integer >= 0)
    Add perturbation steps after each optimization step
    flag: -pert %d
rbn2_val: (a tuple of the form: (a long integer >= 0, a long integer
    >= 0))
    Number of bins in the histogram for reference image for given time
    point
    flag: -rbn %d %d
rbn_val: (a long integer >= 0)
    Number of bins in the histogram for reference image
    flag: --rbn %d
ref_smooth_val: (a float)
    Smoothing kernel width for reference image
    flag: -smooR %f
res_file: (a file name)
    The output resampled image
    flag: -res %s
rlwth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
```

```

        Lower threshold for reference image at the specified time point
        flag: -rLwTh %d %f
rlwth_thr_val: (a float)
        Lower threshold for reference image
        flag: --rLwTh %f
rmask_file: (an existing file name)
        Reference image mask
        flag: -rmask %s
rupth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
        Upper threshold for reference image at the specified time point
        flag: -rUpTh %d %f
rupth_thr_val: (a float)
        Upper threshold for reference image
        flag: --rUpTh %f
smooth_grad_val: (a float)
        Kernel width for smoothing the metric gradient
        flag: -smoothGrad %f
ssd2_flag: (a long integer >= 0)
        Use SSD as the similarity measure for a given time point
        flag: -ssd %d
ssd_flag: (a boolean)
        Use SSD as the similarity measure
        flag: --ssd
sx_val: (a float)
        Final grid spacing along the x axes
        flag: -sx %f
sy_val: (a float)
        Final grid spacing along the y axes
        flag: -sy %f
sz_val: (a float)
        Final grid spacing along the z axes
        flag: -sz %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
vel_flag: (a boolean)
        Use velocity field integration
        flag: -vel
verbosity_off_flag: (a boolean)
        Turn off verbose output
        flag: -voff

```

Outputs:

```

avg_output: (a string)
        Output string in the format for reg_average
cpp_file: (a file name)
        The output CPP file
invcpp_file: (a file name)
        The output inverse CPP file
invres_file: (a file name)
        The output inverse res file
res_file: (a file name)
        The output resampled image

```

interfaces.niftyreg.regutils

RegAverage

[Link to code](#)

Wraps command **reg_average**

Interface for executable reg_average from NiftyReg platform.

Compute average matrix or image from a list of matrices or image. The tool can be use to resample images given input transformation parametrisation as well as to demean transformations in Euclidean or log-Euclidean space.

This interface is different than the others in the way that the options will be written in a command file that is given as a parameter.

[Source code](#)

Examples

```
>>> from nipy.interfaces import niftyreg
>>> node = niftyreg.RegAverage()
>>> one_file = 'im1.nii'
>>> two_file = 'im2.nii'
>>> three_file = 'im3.nii'
>>> node.inputs.avg_files = [one_file, two_file, three_file]
>>> node.cmdline
'reg_average --cmd_file ../reg_average_cmd'
```

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
avg_files: (a list of items which are a file name)
    Averaging of images/affine transformations
    flag: -avg %s, position: 1
    mutually_exclusive: avg_lts_files, avg_ref_file, demean1_ref_file,
        demean2_ref_file, demean3_ref_file, warp_files
avg_lts_files: (a list of items which are a file name)
    Robust average of affine transformations
    flag: -avg_lts %s, position: 1
    mutually_exclusive: avg_files, avg_ref_file, demean1_ref_file,
        demean2_ref_file, demean3_ref_file, warp_files
avg_ref_file: (a file name)
    All input images are resampled into the space of <reference image>
    and averaged. A cubic spline interpolation scheme is used for
    resampling
    flag: -avg_tran %s, position: 1
    mutually_exclusive: avg_files, avg_lts_files, demean1_ref_file,
        demean2_ref_file, demean3_ref_file
    requires: warp_files
demean1_ref_file: (a file name)
    Average images and demean average image that have affine
    transformations to a common space
    flag: -demean1 %s, position: 1
    mutually_exclusive: avg_files, avg_lts_files, avg_ref_file,
        demean2_ref_file, demean3_ref_file
```

```

    requires: warp_files
demean2_ref_file: (a file name)
    Average images and demean average image that have non-rigid
    transformations to a common space
    flag: -demean2 %s, position: 1
    mutually_exclusive: avg_files, avg_lts_files, avg_ref_file,
        demean1_ref_file, demean3_ref_file
    requires: warp_files
demean3_ref_file: (a file name)
    Average images and demean average image that have linear and non-
    rigid transformations to a common space
    flag: -demean3 %s, position: 1
    mutually_exclusive: avg_files, avg_lts_files, avg_ref_file,
        demean1_ref_file, demean2_ref_file
    requires: warp_files
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
omp_core_val: (an integer (int or long), nipy default value: 1)
    Number of openmp thread to use
    flag: -omp %i
out_file: (a file name)
    Output file name
    flag: %s, position: 0
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
warp_files: (a list of items which are a file name)
    transformation files and floating image pairs/triplets to the
    reference space
    flag: %s, position: -1
    mutually_exclusive: avg_files, avg_lts_files

```

Outputs:

```

out_file: (a file name)
    Output file name

```

RegJacobian[Link to code](#)Wraps command **reg_jacobian**

Interface for executable reg_resample from NiftyReg platform.

Tool to generate Jacobian determinant maps from transformation parametrisation generated by reg_f3d

[Source code](#)**Examples**

```

>>> from nipy.interfaces import niftyreg
>>> node = niftyreg.RegJacobian()
>>> node.inputs.ref_file = 'im1.nii'
>>> node.inputs.trans_file = 'warpfield.nii'

```

```
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_jacobian -omp 4 -ref im1.nii -trans warpfield.nii -jac warpfield_jac.nii.gz'
```

Inputs:

```
[Mandatory]
trans_file: (an existing file name)
    The input non-rigid transformation
    flag: -trans %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
omp_core_val: (an integer (int or long), nipy default value: 1)
    Number of openmp thread to use
    flag: -omp %i
out_file: (a file name)
    The output jacobian determinant file name
    flag: %s, position: -1
ref_file: (an existing file name)
    Reference/target file (required if specifying CPP transformations.
    flag: -ref %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
type: ('jac' or 'jacL' or 'jacM', nipy default value: jac)
    Type of jacobian outcome
    flag: -%s, position: -2
```

Outputs:

```
out_file: (a file name)
    The output file
```

RegMeasure

[Link to code](#)

Wraps command **reg_measure**

Interface for executable reg_measure from NiftyReg platform.

Given two input images, compute the specified measure(s) of similarity

[Source code](#)

Examples

```
>>> from nipy.interfaces import niftyreg
>>> node = niftyreg.RegMeasure()
>>> node.inputs.ref_file = 'im1.nii'
```



```
>>> node.inputs.flo_file = 'im2.nii'
>>> node.inputs.measure_type = 'lncc'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_measure -flo im2.nii -lncc -omp 4 -out im2_lncc.txt -ref im1.nii'
```

Inputs:

```
[Mandatory]
flo_file: (an existing file name)
    The input floating/source image
    flag: -flo %s
measure_type: ('ncc' or 'lncc' or 'nmi' or 'ssd')
    Measure of similarity to compute
    flag: -%s
ref_file: (an existing file name)
    The input reference/target image
    flag: -ref %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
omp_core_val: (an integer (int or long), nipyte default value: 1)
    Number of openmp thread to use
    flag: -omp %i
out_file: (a file name)
    The output text file containing the measure
    flag: -out %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
    The output text file containing the measure
```

RegResample

[Link to code](#)Wraps command **reg_resample**

Interface for executable reg_resample from NiftyReg platform.

Tool to resample floating image in the space of a defined reference image given a transformation parametrisation generated by reg_aladin, reg_f3d or reg_transform

[Source code](#)

Examples

```
>>> from nipyre.interfaces import niftyreg
>>> node = niftyreg.RegResample()
>>> node.inputs.ref_file = 'im1.nii'
>>> node.inputs.flo_file = 'im2.nii'
>>> node.inputs.trans_file = 'warpfield.nii'
>>> node.inputs.inter_val = 'LIN'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_resample -flo im2.nii -inter 1 -omp 4 -ref im1.nii -trans warpfield.nii -res_
↳im2_res.nii.gz'
```

Inputs:

```
[Mandatory]
flo_file: (an existing file name)
    The input floating/source image
    flag: -flo %s
ref_file: (an existing file name)
    The input reference/target image
    flag: -ref %s

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inter_val: ('NN' or 'LIN' or 'CUB' or 'SINC')
    Interpolation type
    flag: -inter %d
omp_core_val: (an integer (int or long), nipyre default value: 1)
    Number of openmp thread to use
    flag: -omp %i
out_file: (a file name)
    The output filename of the transformed image
    flag: %s, position: -1
pad_val: (a float)
    Padding value
    flag: -pad %f
psf_alg: (0 or 1)
    Minimise the matrix metric (0) or the determinant (1) when
    estimating the PSF [0]
    flag: -psf_alg %d
psf_flag: (a boolean)
    Perform the resampling in two steps to resample an image to a lower
    resolution
    flag: -psf
tensor_flag: (a boolean)
    Resample Tensor Map
    flag: -tensor
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
```

```

        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
trans_file: (an existing file name)
    The input transformation file
    flag: -trans %s
type: ('res' or 'blank', nipy default value: res)
    Type of output
    flag: -%s, position: -2
verbosity_off_flag: (a boolean)
    Turn off verbose output
    flag: -voff

```

Outputs:

```

out_file: (a file name)
    The output filename of the transformed image

```

RegTools

[Link to code](#)Wraps command **reg_tools**Interface for executable **reg_tools** from NiftyReg platform.

Tool delivering various actions related to registration such as resampling the input image to a chosen resolution or remove the nan and inf in the input image by a specified value.

[Source code](#)

Examples

```

>>> from nipy.interfaces import niftyreg
>>> node = niftyreg.RegTools()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.mul_val = 4
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_tools -in im1.nii -mul 4.0 -omp 4 -out im1_tools.nii.gz'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    The input image file path
    flag: -in %s

[Optional]
add_val: (a float or an existing file name)
    Add to the input image or value
    flag: -add %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bin_flag: (a boolean)
    Binarise the input image
    flag: -bin
chg_res_val: (a tuple of the form: (a float, a float, a float))
    Change the resolution of the input image
    flag: -chgres %f %f %f
div_val: (a float or an existing file name)

```

```

        Divide the input by image or value
        flag: -div %s
down_flag: (a boolean)
        Downsample the image by a factor of 2
        flag: -down
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
iso_flag: (a boolean)
        Make output image isotropic
        flag: -iso
mask_file: (an existing file name)
        Values outside the mask are set to NaN
        flag: -nan %s
mul_val: (a float or an existing file name)
        Multiply the input by image or value
        flag: -mul %s
noscl_flag: (a boolean)
        Set scale, slope to 0 and 1
        flag: -noscl
omp_core_val: (an integer (int or long), nipy default value: 1)
        Number of openmp thread to use
        flag: -omp %i
out_file: (a file name)
        The output file name
        flag: -out %s
rms_val: (an existing file name)
        Compute the mean RMS between the images
        flag: -rms %s
smo_g_val: (a tuple of the form: (a float, a float, a float))
        Smooth the input image using a Gaussian kernel
        flag: -smoG %f %f %f
smo_s_val: (a tuple of the form: (a float, a float, a float))
        Smooth the input image using a cubic spline kernel
        flag: -smoS %f %f %f
sub_val: (a float or an existing file name)
        Add to the input image or value
        flag: -sub %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
thr_val: (a float)
        Binarise the input image with the given threshold
        flag: -thr %f

```

Outputs:

```

out_file: (an existing file name)
        The output file

```

RegTransform[Link to code](#)

Wraps command **reg_transform**

Interface for executable reg_transform from NiftyReg platform.

Tools to convert transformation parametrisation from one type to another as well as to compose, inverse or half transformations.

[Source code](#)

Examples

```
>>> from nipy.interfaces import niftyreg
>>> node = niftyreg.RegTransform()
>>> node.inputs.def_input = 'warpfield.nii'
>>> node.inputs.omp_core_val = 4
>>> node.cmdline
'reg_transform -omp 4 -def warpfield.nii ../warpfield_trans.nii.gz'
```

Inputs:

```
[Mandatory]

[Optional]
aff_2_rig_input: (an existing file name)
    Extract the rigid component from affine transformation
    flag: -aff2rig %s, position: -2
    mutually_exclusive: def_input, disp_input, flow_input, comp_input,
    upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
    make_aff_input, flirt_2_nr_input
args: (a unicode string)
    Additional parameters to the command
    flag: %s
comp_input: (an existing file name)
    compose two transformations
    flag: -comp %s, position: -3
    mutually_exclusive: def_input, disp_input, flow_input,
    upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
    make_aff_input, aff_2_rig_input, flirt_2_nr_input
    requires: comp_input2
comp_input2: (an existing file name)
    compose two transformations
    flag: %s, position: -2
def_input: (an existing file name)
    Compute deformation field from transformation
    flag: -def %s, position: -2
    mutually_exclusive: disp_input, flow_input, comp_input,
    upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
    make_aff_input, aff_2_rig_input, flirt_2_nr_input
disp_input: (an existing file name)
    Compute displacement field from transformation
    flag: -disp %s, position: -2
    mutually_exclusive: def_input, flow_input, comp_input,
    upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
    make_aff_input, aff_2_rig_input, flirt_2_nr_input
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
flirt_2_nr_input: (a tuple of the form: (an existing file name, an
    existing file name, an existing file name))
    Convert a FLIRT affine transformation to niftyreg affine
```

```

transformation
flag: -flirtAff2NR %s %s %s, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
  upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
  make_aff_input, aff_2_rig_input
flow_input: (an existing file name)
  Compute flow field from spline SVF
flag: -flow %s, position: -2
mutually_exclusive: def_input, disp_input, comp_input,
  upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
  make_aff_input, aff_2_rig_input, flirt_2_nr_input
half_input: (an existing file name)
  Half way to the input transformation
flag: -half %s, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
  upd_s_form_input, inv_aff_input, inv_nrr_input, make_aff_input,
  aff_2_rig_input, flirt_2_nr_input
ignore_exception: (a boolean, nipy default value: False)
  Print an error message instead of throwing an exception in case the
  interface fails to run
inv_aff_input: (an existing file name)
  Invert an affine transformation
flag: -invAff %s, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
  upd_s_form_input, inv_nrr_input, half_input, make_aff_input,
  aff_2_rig_input, flirt_2_nr_input
inv_nrr_input: (a tuple of the form: (an existing file name, an
  existing file name))
  Invert a non-linear transformation
flag: -invNrr %s %s, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
  upd_s_form_input, inv_aff_input, half_input, make_aff_input,
  aff_2_rig_input, flirt_2_nr_input
make_aff_input: (a tuple of the form: (a float, a float, a float, a
  float, a float, a float, a float, a float, a float, a
  float, a float))
  Make an affine transformation matrix
flag: -makeAff %f %f %f %f %f %f %f %f %f %f %f %f, position: -2
mutually_exclusive: def_input, disp_input, flow_input, comp_input,
  upd_s_form_input, inv_aff_input, inv_nrr_input, half_input,
  aff_2_rig_input, flirt_2_nr_input
omp_core_val: (an integer (int or long), nipy default value: 1)
  Number of openmp thread to use
flag: -omp %i
out_file: (a file name)
  transformation file to write
flag: %s, position: -1
ref1_file: (an existing file name)
  The input reference/target image
flag: -ref %s, position: 0
ref2_file: (an existing file name)
  The input second reference/target image
flag: -ref2 %s, position: 1
requires: ref1_file
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
  Control terminal output: `stream` - displays to terminal immediately
  (default), `allatonce` - waits till command is finished to display
  output, `file` - writes output to file, `none` - output is ignored

```

```
upd_s_form_input: (an existing file name)
    Update s-form using the affine transformation
    flag: -updSform %s, position: -3
    mutually_exclusive: def_input, disp_input, flow_input, comp_input,
        inv_aff_input, inv_nrr_input, half_input, make_aff_input,
        aff_2_rig_input, flirt_2_nr_input
    requires: upd_s_form_input2
upd_s_form_input2: (an existing file name)
    Update s-form using the affine transformation
    flag: %s, position: -2
    requires: upd_s_form_input
```

Outputs:

```
out_file: (a file name)
    Output File (transformation in any format)
```


interfaces.niftyseg.base

NiftySegCommand

[Link to code](#)

Wraps command **None**

Base support interface for NiftySeg commands.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

None

interfaces.niftyseg.em

EM

[Link to code](#)

Wraps command `seg_EM`

Interface for executable `seg_EM` from NiftySeg platform.

`seg_EM` is a general purpose intensity based image segmentation tool. In it's simplest form, it takes in one 2D or 3D image and segments it in `n` classes.

[Source code](#) | [Documentation](#)

Examples

```
>>> from nipy.interfaces import niftyseg
>>> node = niftyseg.EM()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.no_prior = 4
>>> node.cmdline
'seg_EM -in im1.nii -nopriors 4 -bc_out im1_bc_em.nii.gz -out im1_em.nii.gz -out_
↪outlier im1_outlier_em.nii.gz'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    Input image to segment
    flag: -in %s, position: 4
no_prior: (an integer (int or long))
    Number of classes to use without prior
    flag: -nopriors %s
    mutually_exclusive: prior_4D, priors
prior_4D: (an existing file name)
    4D file containing the priors
    flag: -prior4D %s
    mutually_exclusive: no_prior, priors
priors: (a list of items which are any value)
    List of priors filepaths.
    flag: %s
    mutually_exclusive: no_prior, prior_4D

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bc_order_val: (an integer (int or long))
    Polynomial order for the bias field
    flag: -bc_order %s
bc_thresh_val: (a float)
    Bias field correction will run only if the ratio of improvement is
    below bc_thresh. (default=0 [OFF])
    flag: -bc_thresh %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_file: (an existing file name)
    Filename of the ROI for label fusion
    flag: -mask %s
max_iter: (an integer (int or long))
    Maximum number of iterations
```

```

        flag: -max_iter %s
min_iter: (an integer (int or long))
        Minimum number of iterations
        flag: -min_iter %s
mrf_beta_val: (a float)
        Weight of the Markov Random Field
        flag: -mrf_beta %s
out_bc_file: (a file name)
        Output bias corrected image
        flag: -bc_out %s
out_file: (a file name)
        Output segmentation
        flag: -out %s
out_outlier_file: (a file name)
        Output outlieriness image
        flag: -out_outlier %s
outlier_val: (a tuple of the form: (a float, a float))
        Outlier detection as in (Van Leemput TMI 2003). <fl1> is the
        Mahalanobis threshold [recommended between 3 and 7] <fl2> is a
        convergence ratio below which the outlier detection is going to be
        done [recommended 0.01]
        flag: -outlier %s %s
reg_val: (a float)
        Amount of regularization over the diagonal of the covariance matrix
        [above 1]
        flag: -reg %s
relax_priors: (a tuple of the form: (a float, a float))
        Relax Priors [relaxation factor: 0<rf<1 (recommended=0.5), gaussian
        regularization: gstd>0 (recommended=2.0)] /only 3D/
        flag: -rf %s %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_bc_file: (a file name)
        Output bias corrected image
out_file: (a file name)
        Output segmentation
out_outlier_file: (a file name)
        Output outlieriness image

```

interfaces.niftyseg.label_fusion

CalcTopNCC

[Link to code](#)Wraps command **seg_CalcTopNCC**

Interface for executable seg_CalcTopNCC from NiftySeg platform.

Examples

```

>>> from nipy.interfaces import niftyseg
>>> node = niftyseg.CalcTopNCC()

```

```
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.num_templates = 2
>>> node.inputs.in_templates = ['im2.nii', 'im3.nii']
>>> node.inputs.top_templates = 1
>>> node.cmdline
'seg_CalcTopNCC -target im1.nii -templates 2 im2.nii im3.nii -n 1'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    Target file
    flag: -target %s, position: 1
in_templates: (a list of items which are an existing file name)
    flag: %s, position: 3
num_templates: (an integer (int or long))
    Number of Templates
    flag: -templates %s, position: 2
top_templates: (an integer (int or long))
    Number of Top Templates
    flag: -n %s, position: 4

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask_file: (an existing file name)
    Filename of the ROI for label fusion
    flag: -mask %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_files: (any value)
```

LabelFusion

[Link to code](#)

Wraps command **seg_LabFusion**

Interface for executable seg_LabelFusion from NiftySeg platform using type STEPS as classifier Fusion.

This executable implements 4 fusion strategies (-STEPS, -STAPLE, -MV or -SBA), all of them using either a global (-GNCC), ROI-based (-ROINCC), local (-LNCC) or no image similarity (-ALL). Combinations of fusion algorithms and similarity metrics give rise to different variants of known algorithms. As an example, using LNCC and MV as options will run a locally weighted voting strategy with LNCC derived weights, while using STAPLE and LNCC is equivalent to running STEPS as per its original formulation. A few other options pertaining the use of an MRF (-MRF beta), the initial sensitivity and specificity estimates and the use of only non-consensus voxels (-unc) for the STAPLE and STEPS algorithm. All processing can be masked (-mask), greatly reducing memory consumption.

As an example, the command to use STEPS should be: `seg_LabFusion -in 4D_Propragated_Labels_to_fuse.nii -out FusedSegmentation.nii -STEPS 2 15 TargetImage.nii 4D_Propagated_Intensities.nii`

[Source code](#) | [Documenta­tion](#)

Examples

```
>>> from nipy.interfaces import niftyseg
>>> node = niftyseg.LabelFusion()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.kernel_size = 2.0
>>> node.inputs.file_to_seg = 'im2.nii'
>>> node.inputs.template_file = 'im3.nii'
>>> node.inputs.template_num = 2
>>> node.inputs.classifier_type = 'STEPS'
>>> node.cmdline
'seg_LabFusion -in im1.nii -STEPS 2.000000 2 im2.nii im3.nii -out im1_steps.nii'
```

Inputs:

```
[Mandatory]
classifier_type: ('STEPS' or 'STAPLE' or 'MV' or 'SBA')
    Type of Classifier Fusion.
    flag: -%s, position: 2
file_to_seg: (an existing file name)
    Original image to segment (3D Image)
in_file: (an existing file name)
    Filename of the 4D integer label image.
    flag: -in %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
conv: (a float)
    Ratio for convergence (default epsilon = 10^-5).
    flag: -conv %f
dilation_roi: (an integer (int or long))
    Dilation of the ROI ( <int> d>=1 )
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
kernel_size: (a float)
    Gaussian kernel size in mm to compute the local similarity
mask_file: (an existing file name)
    Filename of the ROI for label fusion
    flag: -mask %s
max_iter: (an integer (int or long))
    Maximum number of iterations (default = 15).
    flag: -max_iter %d
mrf_value: (a float)
    MRF prior strength (between 0 and 5)
    flag: -MRF_beta %f
out_file: (a file name)
    Output consensus segmentation
```

```

        flag: -out %s
prob_flag: (a boolean)
    Probabilistic/Fuzzy segmented image
    flag: -outProb
prob_update_flag: (a boolean)
    Update label proportions at each iteration
    flag: -prop_update
proportion: (a float)
    Proportion of the label (only for single labels).
    flag: -prop %s
set_pq: (a tuple of the form: (a float, a float))
    Value of P and Q [ 0 < (P,Q) < 1 ] (default = 0.99 0.99)
    flag: -setPQ %f %f
sm_ranking: ('ALL' or 'GNCC' or 'ROINCC' or 'LNCC', nipype default
    value: ALL)
    Ranking for STAPLE and MV
    flag: -%s, position: 3
template_file: (an existing file name)
    Registered templates (4D Image)
template_num: (an integer (int or long))
    Number of labels to use
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
unc: (a boolean)
    Only consider non-consensus voxels to calculate statistics
    flag: -unc
unc_thresh: (a float)
    If <float> percent of labels agree, then area is not uncertain.
    flag: -uncthres %f
verbose: ('0' or '1' or '2')
    Verbose level [0 = off, 1 = on, 2 = debug] (default = 0)
    flag: -v %s

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

interfaces.niftyseg.lesions

FillLesions

[Link to code](#)Wraps command `seg_FillLesions`Interface for executable `seg_FillLesions` from NiftySeg platform.

Fill all the masked lesions with WM intensity average.

[Source code](#) | [Documentation](#)

Examples

```

>>> from nipype.interfaces import niftyseg
>>> node = niftyseg.FillLesions()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.lesion_mask = 'im2.nii'

```

```
>>> node.cmdline
'seg_FillLesions -i im1.nii -l im2.nii -o im1_lesions_filled.nii.gz'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    Input image to fill lesions
    flag: -i %s, position: 1
lesion_mask: (an existing file name)
    Lesion mask
    flag: -l %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bin_mask: (a file name)
    Give a binary mask with the valid search areas.
    flag: -mask %s
cwf: (a float)
    Patch cardinality weighting factor (by default 2).
    flag: -cwf %f
debug: (a boolean)
    Save all intermidium files (by default OFF).
    flag: -debug
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_dilation: (an integer (int or long))
    Dilate the mask <int> times (in voxels, by default 0)
    flag: -dil %d
match: (a float)
    Percentage of minimum number of voxels between patches <float> (by
    default 0.5).
    flag: -match %f
other: (a boolean)
    Guizard et al. (FIN 2015) method, it doesn't include the
    multiresolution/hierarchical inpainting part, this part needs to be
    done with some external software such as reg_tools and reg_resample
    from NiftyReg. By default it uses the method presented in Prados et
    al. (Neuroimage 2016).
    flag: -other
out_datatype: (a string)
    Set output <datatype> (char, short, int, uchar, ushort, uint, float,
    double).
    flag: -odt %s
out_file: (a file name)
    The output filename of the fill lesions results
    flag: -o %s, position: 3
search: (a float)
    Minimum percentage of valid voxels in target patch <float> (by
    default 0).
    flag: -search %f
```

```

size: (an integer (int or long))
    Search regions size respect biggest patch size (by default 4).
    flag: -size %d
smooth: (a float)
    Smoothing by <float> (in minimal 6-neighbourhood voxels (by default
    0.1)).
    flag: -smo %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
use_2d: (a boolean)
    Uses 2D patches in the Z axis, by default 3D.
    flag: -2D
verbose: (a boolean)
    Verbose (by default OFF).
    flag: -v

```

Outputs:

```

out_file: (a file name)
    Output segmentation

```

interfaces.niftyseg.maths

BinaryMaths

[Link to code](#)Wraps command **seg_maths**

Interface for executable seg_maths from NiftySeg platform.

Interface to use any binary mathematical operations that can be performed with the seg_maths command-line program.

See below for those operations:

```

mul - <float/file> - Multiply image <float> value or by other image.
div - <float/file> - Divide image by <float> or by other image.
add - <float/file> - Add image by <float> or by other image.
sub - <float/file> - Subtract image by <float> or by other image.
pow - <float> - Image to the power of <float>.
thr - <float> - Threshold the image below <float>.
uthr - <float> - Threshold image above <float>.
smo - <float> - Gaussian smoothing by std <float> (in voxels and up to 4-D).
edge - <float> - Calculate the edges of the image using a threshold <float>.
sobel3 - <float> - Calculate the edges of all timepoints using a Sobel filter with a 3x3x3 kernel and applying
<float> gaussian smoothing.
sobel5 - <float> - Calculate the edges of all timepoints using a Sobel filter with a 5x5x5 kernel and applying
<float> gaussian smoothing.
min - <file> - Get the min per voxel between <current> and <file>.
smol - <float> - Gaussian smoothing of a 3D label image.
geo - <float/file> - Geodesic distance according to the speed function <float/file>
llsnorm <file_norm> - Linear LS normalisation between current and <file_norm>
masknan <file_norm> - Assign everything outside the mask (mask==0) with NaNs
hdr_copy <file> - Copy header from working image to <file> and save in <output>.
splitinter <x/y/z> - Split interleaved slices in direction <x/y/z> into separate time points

```

[Source code](#) | [Documentation](#)

Examples

```
>>> import copy
>>> from nipy.interfaces import niftyseg
>>> binary = niftyseg.BinaryMaths()
>>> binary.inputs.in_file = 'im1.nii'
>>> binary.inputs.output_datatype = 'float'
>>> # Test sub operation
>>> binary_sub = copy.deepcopy(binary)
>>> binary_sub.inputs.operation = 'sub'
>>> binary_sub.inputs.operand_file = 'im2.nii'
>>> binary_sub.cmdline
'seg_maths im1.nii -sub im2.nii -odt float im1_sub.nii'
>>> binary_sub.run()
>>> # Test mul operation
>>> binary_mul = copy.deepcopy(binary)
>>> binary_mul.inputs.operation = 'mul'
>>> binary_mul.inputs.operand_value = 2.0
>>> binary_mul.cmdline
'seg_maths im1.nii -mul 2.00000000 -odt float im1_mul.nii'
>>> binary_mul.run()
>>> # Test llsnorm operation
>>> binary_llsnorm = copy.deepcopy(binary)
>>> binary_llsnorm.inputs.operation = 'llsnorm'
>>> binary_llsnorm.inputs.operand_file = 'im2.nii'
>>> binary_llsnorm.cmdline
'seg_maths im1.nii -llsnorm im2.nii -odt float im1_llsnorm.nii'
>>> binary_llsnorm.run()
>>> # Test splitinter operation
>>> binary_splitinter = copy.deepcopy(binary)
>>> binary_splitinter.inputs.operation = 'splitinter'
>>> binary_splitinter.inputs.operand_str = 'z'
>>> binary_splitinter.cmdline
'seg_maths im1.nii -splitinter z -odt float im1_splitinter.nii'
>>> binary_splitinter.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2
operand_file: (an existing file name)
        second image to perform operation with
        flag: %s, position: 5
        mutually_exclusive: operand_value, operand_str
operand_str: ('x' or 'y' or 'z')
        string value to perform operation splitinter
        flag: %s, position: 5
        mutually_exclusive: operand_value, operand_file
operand_value: (a float)
        float value to perform operation with
        flag: %.8f, position: 5
        mutually_exclusive: operand_file, operand_str
operation: ('mul' or 'div' or 'add' or 'sub' or 'pow' or 'thr' or
            'uthr' or 'smo' or 'edge' or 'sobel3' or 'sobel5' or 'min' or
            'smol' or 'geo' or 'llsnorm' or 'masknan' or 'hdr_copy' or
            'splitinter')
        operation to perform
```

```

        flag: -%s, position: 4

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    image to write
    flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    flag: -odt %s, position: -3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
    image written after calculations

```

BinaryMathsInteger[Link to code](#)Wraps command **seg_maths**

Interface for executable seg_maths from NiftySeg platform.

Interface to use any integer mathematical operations that can be performed with the seg_maths command-line program.

See below for those operations:: (requiring integer values)

equal - <int> - Get voxels equal to <int>

dil - <int> - Dilate the image <int> times (in voxels).

ero - <int> - Erode the image <int> times (in voxels).

tp - <int> - Extract time point <int>

crop - <int> - Crop <int> voxels around each 3D volume.

pad - <int> - Pad <int> voxels with NaN value around each 3D volume.

[Source code](#) | [Documentation](#)**Examples**

```

>>> import copy
>>> from nipy.interfaces.niftyseg import BinaryMathsInteger
>>> binaryi = BinaryMathsInteger()
>>> binaryi.inputs.in_file = 'im1.nii'
>>> binaryi.inputs.output_datatype = 'float'
>>> # Test dil operation
>>> binaryi_dil = copy.deepcopy(binaryi)
>>> binaryi_dil.inputs.operation = 'dil'

```

```

>>> binaryi_dil.inputs.operand_value = 2
>>> binaryi_dil.cmdline
'seg_maths im1.nii -dil 2 -odt float im1_dil.nii'
>>> binaryi_dil.run()
>>> # Test dil operation
>>> binaryi_ero = copy.deepcopy(binaryi)
>>> binaryi_ero.inputs.operation = 'ero'
>>> binaryi_ero.inputs.operand_value = 1
>>> binaryi_ero.cmdline
'seg_maths im1.nii -ero 1 -odt float im1_ero.nii'
>>> binaryi_ero.run()
>>> # Test pad operation
>>> binaryi_pad = copy.deepcopy(binaryi)
>>> binaryi_pad.inputs.operation = 'pad'
>>> binaryi_pad.inputs.operand_value = 4
>>> binaryi_pad.cmdline
'seg_maths im1.nii -pad 4 -odt float im1_pad.nii'
>>> binaryi_pad.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2
operand_value: (an integer (int or long))
        int value to perform operation with
        flag: %d, position: 5
operation: ('dil' or 'ero' or 'tp' or 'equal' or 'pad' or 'crop')
        operation to perform
        flag: -%s, position: 4

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
out_file: (a file name)
        image to write
        flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for output (default uses input type)
        flag: -odt %s, position: -3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
        image written after calculations

```

MathsCommand

[Link to code](#)

Wraps command **seg_maths**

Base Command Interface for seg_maths interfaces.

The executable seg_maths enables the sequential execution of arithmetic operations, like multiplication (-mul), division (-div) or addition (-add), binarisation (-bin) or thresholding (-thr) operations and convolution by a Gaussian kernel (-smo). It also allows mathematical morphology based operations like dilation (-dil), erosion (-ero), connected components (-lconcomp) and hole filling (-fill), Euclidean (-euc) and geodesic (-geo) distance transforms, local image similarity metric calculation (-lncc and -lssd). Finally, it allows multiple operations over the dimensionality of the image, from merging 3D images together as a 4D image (-merge) or splitting (-split or -tp) 4D images into several 3D images, to estimating the maximum, minimum and average over all time-points, etc.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          image to write
          flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                  datatype to use for output (default uses input type)
                  flag: -odt %s, position: -3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          image written after calculations
```

Merge

[Link to code](#)

Wraps command **seg_maths**

Interface for executable seg_maths from NiftySeg platform.

Interface to use the merge operation that can be performed with the seg_maths command-line program.

See below for this option:

merge <i> <d> <files> Merge <i> images and the working image in the <d> dimension

Source code | Documentation

Examples

```
>>> from nipyre.interfaces import niftyseg
>>> node = niftyseg.Merge()
>>> node.inputs.in_file = 'im1.nii'
>>> files = ['im2.nii', 'im3.nii']
>>> node.inputs.merge_files = files
>>> node.inputs.dimension = 2
>>> node.inputs.output_datatype = 'float'
>>> node.cmdline
'seg_maths im1.nii -merge 2 2 im2.nii im3.nii -odt float im1_merged.nii'
```

Inputs:

```
[Mandatory]
dimension: (an integer (int or long))
    Dimension to merge the images.
in_file: (an existing file name)
    image to operate on
    flag: %s, position: 2
merge_files: (a list of items which are an existing file name)
    List of images to merge to the working image <input>.
    flag: %s, position: 4

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    image to write
    flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    flag: -odt %s, position: -3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
    image written after calculations
```

TupleMaths

[Link to code](#)

Wraps command `seg_maths`

Interface for executable `seg_maths` from NiftySeg platform.

Interface to use any tuple mathematical operations that can be performed with the `seg_maths` command-line program.

See below for those operations:

`lncc <file> <std>` Local CC between current img and `<file>` on a kernel with `<std>`

`lssd <file> <std>` Local SSD between current img and `<file>` on a kernel with `<std>`

`lltsnorm <file_norm> <float>` Linear LTS normalisation assuming `<float>` percent outliers

[Source code](#) | [Documentation](#)

Examples

```
>>> import copy
>>> from nipyype.interfaces import niftyseg
>>> tuple = niftyseg.TupleMaths()
>>> tuple.inputs.in_file = 'im1.nii'
>>> tuple.inputs.output_datatype = 'float'
```

```
>>> # Test lncc operation
>>> tuple_lncc = copy.deepcopy(tuple)
>>> tuple_lncc.inputs.operation = 'lncc'
>>> tuple_lncc.inputs.operand_file1 = 'im2.nii'
>>> tuple_lncc.inputs.operand_value2 = 2.0
>>> tuple_lncc.cmdline
'seg_maths im1.nii -lncc im2.nii 2.00000000 -odt float im1_lncc.nii'
>>> tuple_lncc.run()
```

```
>>> # Test lssd operation
>>> tuple_lssd = copy.deepcopy(tuple)
>>> tuple_lssd.inputs.operation = 'lssd'
>>> tuple_lssd.inputs.operand_file1 = 'im2.nii'
>>> tuple_lssd.inputs.operand_value2 = 1.0
>>> tuple_lssd.cmdline
'seg_maths im1.nii -lssd im2.nii 1.00000000 -odt float im1_lssd.nii'
>>> tuple_lssd.run()
```

```
>>> # Test lltsnorm operation
>>> tuple_lltsnorm = copy.deepcopy(tuple)
>>> tuple_lltsnorm.inputs.operation = 'lltsnorm'
>>> tuple_lltsnorm.inputs.operand_file1 = 'im2.nii'
>>> tuple_lltsnorm.inputs.operand_value2 = 0.01
>>> tuple_lltsnorm.cmdline
'seg_maths im1.nii -lltsnorm im2.nii 0.01000000 -odt float im1_lltsnorm.nii'
>>> tuple_lltsnorm.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         image to operate on
         flag: %s, position: 2
operand_file1: (an existing file name)
               image to perform operation 1 with
               flag: %s, position: 5
               mutually_exclusive: operand_value1
operand_file2: (an existing file name)
               image to perform operation 2 with
```

```

        flag: %s, position: 6
        mutually_exclusive: operand_value2
operand_value1: (a float)
        float value to perform operation 1 with
        flag: %.8f, position: 5
        mutually_exclusive: operand_file1
operand_value2: (a float)
        float value to perform operation 2 with
        flag: %.8f, position: 6
        mutually_exclusive: operand_file2
operation: ('lncc' or 'lssd' or 'lltsnorm')
        operation to perform
        flag: -%s, position: 4

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
out_file: (a file name)
        image to write
        flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for output (default uses input type)
        flag: -odt %s, position: -3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
        image written after calculations

```

UnaryMaths[Link to code](#)**Wraps command `seg_maths`**Interface for executable `seg_maths` from NiftySeg platform.Interface to use any unary mathematical operations that can be performed with the `seg_maths` command-line program.

See below for those operations:

`sqrt` - Square root of the image).`exp` - Exponential root of the image.`log` - Log of the image.`recip` - Reciprocal (1/I) of the image.`abs` - Absolute value of the image.`bin` - Binarise the image.`otsu` - Otsu thresholding of the current image.

lconcomp - Take the largest connected component
concomp6 - Label the different connected components with a 6NN kernel
concomp26 - Label the different connected components with a 26NN kernel
fill - Fill holes in binary object (e.g. fill ventricle in brain mask).
euc - Euclidean distance transform
tpmax - Get the time point with the highest value (binarise 4D probabilities)
tmean - Mean value of all time points.
tmax - Max value of all time points.
tmin - Mean value of all time points.
splitlab - Split the integer labels into multiple timepoints
removenan - Remove all NaNs and replace them with 0
isnan - Binary image equal to 1 if the value is NaN and 0 otherwise
subsamp2 - Subsample the image by 2 using NN sampling (qform and sform scaled)
scl - Reset scale and slope info.
4to5 - Flip the 4th and 5th dimension.
range - Reset the image range to the min max.
[Source code](#) | [Documentation](#)

Examples

```
>>> import copy
>>> from nipy.interfaces import niftyseg
>>> unary = niftyseg.UnaryMaths()
>>> unary.inputs.output_datatype = 'float'
>>> unary.inputs.in_file = 'im1.nii'
>>> # Test sqrt operation
>>> unary_sqrt = copy.deepcopy(unary)
>>> unary_sqrt.inputs.operation = 'sqrt'
>>> unary_sqrt.cmdline
'seg_maths im1.nii -sqrt -odt float im1_sqrt.nii'
>>> unary_sqrt.run()
>>> # Test sqrt operation
>>> unary_abs = copy.deepcopy(unary)
>>> unary_abs.inputs.operation = 'abs'
>>> unary_abs.cmdline
'seg_maths im1.nii -abs -odt float im1_abs.nii'
>>> unary_abs.run()
>>> # Test bin operation
>>> unary_bin = copy.deepcopy(unary)
>>> unary_bin.inputs.operation = 'bin'
>>> unary_bin.cmdline
'seg_maths im1.nii -bin -odt float im1_bin.nii'
>>> unary_bin.run()
>>> # Test otsu operation
>>> unary_otsu = copy.deepcopy(unary)
>>> unary_otsu.inputs.operation = 'otsu'
>>> unary_otsu.cmdline
'seg_maths im1.nii -otsu -odt float im1_otsu.nii'
>>> unary_otsu.run()
>>> # Test isnan operation
>>> unary_isnan = copy.deepcopy(unary)
>>> unary_isnan.inputs.operation = 'isnan'
>>> unary_isnan.cmdline
'seg_maths im1.nii -isnan -odt float im1_isnan.nii'
>>> unary_isnan.run()
```

Inputs:


```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2
operation: ('sqrt' or 'exp' or 'log' or 'recip' or 'abs' or 'bin' or
           'otsu' or 'lconcomp' or 'concomp6' or 'concomp26' or 'fill' or
           'euc' or 'tpmax' or 'tmean' or 'tmax' or 'tmin' or 'splitlab' or
           'removenan' or 'isnan' or 'subsamp2' or 'scl' or '4to5' or 'range')
        operation to perform
        flag: -%s, position: 4

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipyype default value: {})
        Environment variables
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          image to write
          flag: %s, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
                 datatype to use for output (default uses input type)
                 flag: -odt %s, position: -3
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          image written after calculations
```

interfaces.niftyseg.patchmatch

PatchMatch

[Link to code](#)Wraps command **seg_PatchMatch**

Interface for executable seg_PatchMatch from NiftySeg platform.

The database file is a text file and in each line we have a template file, a mask with the search region to consider and a file with the label to propagate.

Input image, input mask, template images from database and masks from database must have the same 4D resolution (same number of XxYxZ voxels, modalities and/or time-points). Label files from database must have the same 3D resolution (XxYxZ voxels) than input image but can have different number of volumes than the input image allowing to propagate multiple labels in the same execution.

[Source code](#) | [Documentation](#)

Examples

```
>>> from nipy.interfaces import niftyseg
>>> node = niftyseg.PatchMatch()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.mask_file = 'im2.nii'
>>> node.inputs.database_file = 'db.xml'
>>> node.cmdline
'seg_PatchMatch -i im1.nii -m im2.nii -db db.xml -o im1_pm.nii.gz'
```

Inputs:

```
[Mandatory]
database_file: (an existing file name)
    Database with the segmentations
    flag: -db %s, position: 3
in_file: (an existing file name)
    Input image to segment
    flag: -i %s, position: 1
mask_file: (an existing file name)
    Input mask for the area where applies PatchMatch
    flag: -m %s, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cs_size: (an integer (int or long))
    Constrained search area size, number of times bigger than the
    patchsize
    flag: -cs %i
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
it_num: (an integer (int or long))
    Number of iterations for the patchmatch algorithm
    flag: -it %i
match_num: (an integer (int or long))
    Number of better matching
    flag: -match %i
out_file: (a file name)
    The output filename of the patchmatch results
    flag: -o %s, position: 4
patch_size: (an integer (int or long))
    Patch size, #voxels
    flag: -size %i
pm_num: (an integer (int or long))
    Number of patchmatch executions
    flag: -pm %i
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (a file name)
          Output segmentation
```

interfaces.niftyseg.stats

BinaryStats

[Link to code](#)

Wraps command **seg_stats**

Interface for executable seg_stats from NiftySeg platform.

Interface to use any binary statistical operations that can be performed with the seg_stats command-line program.

See below for those operations:

p - <float> - The <float>th percentile of all voxels intensity (float=[0,100])

sa - <ax> - Average of all voxels

ss - <ax> - Standard deviation of all voxels

svp - <ax> - Volume of all probabilistic voxels (sum(<in>) * <volume per voxel>)

al - <in2> - Average value in <in> for each label in <in2>

d - <in2> - Calculate the Dice score between all classes in <in> and <in2>

ncc - <in2> - Normalized cross correlation between <in> and <in2>

nmi - <in2> - Normalized Mutual Information between <in> and <in2>

Vl - <csv> - Volume of each integer label <in>. Save to <csv>file.

Nl - <csv> - Count of each label <in>. Save to <csv> file.

[Source code](#) | [Documentation](#)

Examples

```
>>> import copy
>>> from nipy.interfaces import niftyseg
>>> binary = niftyseg.BinaryStats()
>>> binary.inputs.in_file = 'im1.nii'
>>> # Test sa operation
>>> binary_sa = copy.deepcopy(binary)
>>> binary_sa.inputs.operation = 'sa'
>>> binary_sa.inputs.operand_value = 2.0
>>> binary_sa.cmdline
'seg_stats im1.nii -sa 2.00000000'
>>> binary_sa.run()
>>> # Test ncc operation
>>> binary_ncc = copy.deepcopy(binary)
>>> binary_ncc.inputs.operation = 'ncc'
>>> binary_ncc.inputs.operand_file = 'im2.nii'
>>> binary_ncc.cmdline
'seg_stats im1.nii -ncc im2.nii'
>>> binary_ncc.run()
>>> # Test Nl operation
>>> binary_nl = copy.deepcopy(binary)
>>> binary_nl.inputs.operation = 'Nl'
>>> binary_nl.inputs.operand_file = 'output.csv'
>>> binary_nl.cmdline
'seg_stats im1.nii -Nl output.csv'
>>> binary_nl.run()
```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    image to operate on
    flag: %s, position: 2
operand_file: (an existing file name)
    second image to perform operation with
    flag: %s, position: 5
    mutually_exclusive: operand_value
operand_value: (a float)
    value to perform operation with
    flag: %.8f, position: 5
    mutually_exclusive: operand_file
operation: ('p' or 'sa' or 'ss' or 'svp' or 'al' or 'd' or 'ncc' or
    'nmi' or 'Vl' or 'Nl')
    operation to perform
    flag: -%s, position: 4

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
larger_voxel: (a float)
    Only estimate statistics if voxel is larger than <float>
    flag: -t %f, position: -3
mask_file: (an existing file name)
    statistics within the masked area
    flag: -m %s, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

output: (an array)
    Output array from seg_stats

```

StatsCommand[Link to code](#)Wraps command **seg_stats**

Base Command Interface for seg_stats interfaces.

The executable **seg_stats** enables the estimation of image statistics on continuous voxel intensities (average, standard deviation, min/max, robust range, percentiles, sum, probabilistic volume, entropy, etc) either over the full image or on a per slice basis (slice axis can be specified), statistics over voxel coordinates (location of max, min and centre of mass, bounding box, etc) and statistics over categorical images (e.g. per region volume, count, average, Dice scores, etc). These statistics are robust to the presence of NaNs, and can be constrained by a mask and/or thresholded at a certain level.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
larger_voxel: (a float)
              Only estimate statistics if voxel is larger than <float>
              flag: -t %f, position: -3
mask_file: (an existing file name)
            statistics within the masked area
            flag: -m %s, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
output: (an array)
        Output array from seg_stats
```

UnaryStats

[Link to code](#)**Wraps command `seg_stats`**Interface for executable `seg_stats` from NiftySeg platform.Interface to use any unary statistical operations that can be performed with the `seg_stats` command-line program.

See below for those operations:

r - The range <min max> of all voxels.

R - The robust range (assuming 2% outliers on both sides) of all voxels

a - Average of all voxels

s - Standard deviation of all voxels

v - Volume of all voxels above 0 (<# voxels> * <volume per voxel>)

vl - Volume of each integer label (<# voxels per label> * <volume per voxel>)

vp - Volume of all probabilistic voxels (sum(<in>) * <volume per voxel>)

n - Count of all voxels above 0 (<# voxels>)

np - Sum of all fuzzy voxels (sum(<in>))

e - Entropy of all voxels

ne - Normalized entropy of all voxels

x - Location (i j k x y z) of the smallest value in the image

X - Location (i j k x y z) of the largest value in the image

c - Location (i j k x y z) of the centre of mass of the object

B - Bounding box of all nonzero voxels [xmin xsize ymin ysize zmin zsize]

xvox - Output the number of voxels in the x direction. Replace x with y/z for other directions.

x_{dim} - Output the voxel dimension in the x direction. Replace x with y/z for other directions.

[Source code](#) | [Documentation](#)

Examples

```
>>> import copy
>>> from nipy.interfaces import niftyseg
>>> unary = niftyseg.UnaryStats()
>>> unary.inputs.in_file = 'im1.nii'
>>> # Test v operation
>>> unary_v = copy.deepcopy(unary)
>>> unary_v.inputs.operation = 'v'
>>> unary_v.cmdline
'seg_stats im1.nii -v'
>>> unary_v.run()
>>> # Test vl operation
>>> unary_vl = copy.deepcopy(unary)
>>> unary_vl.inputs.operation = 'vl'
>>> unary_vl.cmdline
'seg_stats im1.nii -vl'
>>> unary_vl.run()
>>> # Test x operation
>>> unary_x = copy.deepcopy(unary)
>>> unary_x.inputs.operation = 'x'
>>> unary_x.cmdline
'seg_stats im1.nii -x'
>>> unary_x.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        flag: %s, position: 2
operation: ('r' or 'R' or 'a' or 's' or 'v' or 'vl' or 'vp' or 'n' or
           'np' or 'e' or 'ne' or 'x' or 'X' or 'c' or 'B' or 'xvox' or
           'xdim')
        operation to perform
        flag: -%s, position: 4

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
larger_voxel: (a float)
              Only estimate statistics if voxel is larger than <float>
              flag: -t %f, position: -3
mask_file: (an existing file name)
            statistics within the masked area
            flag: -m %s, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
```

```
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
output: (an array)
        Output array from seg_stats
```


interfaces.nipy.model

EstimateContrast

[Link to code](#)

Estimate contrast of a fitted model.

Inputs:

```
[Mandatory]
axis: (any value)
beta: (an existing file name)
    beta coefficients of the fitted model
constants: (any value)
contrasts: (a list of items which are a tuple of the form: (a unicode
    string, 'T', a list of items which are a unicode string, a list of
    items which are a float) or a tuple of the form: (a unicode string,
    'T', a list of items which are a unicode string, a list of items
    which are a float, a list of items which are a float) or a tuple of
    the form: (a unicode string, 'F', a list of items which are a tuple
    of the form: (a unicode string, 'T', a list of items which are a
    unicode string, a list of items which are a float) or a tuple of
    the form: (a unicode string, 'T', a list of items which are a
    unicode string, a list of items which are a float, a list of items
    which are a float)))
    List of contrasts with each contrast being a list of the form:
    [('name', 'stat', [condition list], [weight list], [session
    list]]). if
    session list is None or not provided, all sessions are used. For F
    contrasts, the condition list should contain previously defined
    T-contrasts.
dof: (any value)
    degrees of freedom
nvbeta: (any value)
reg_names: (a list of items which are any value)
s2: (an existing file name)
    squared variance of the residuals
```

```
[Optional]
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (a file name)
```

Outputs:

```
p_maps: (a list of items which are an existing file name)
stat_maps: (a list of items which are an existing file name)
z_maps: (a list of items which are an existing file name)
```

FitGLM

[Link to code](#)

Fit GLM model based on the specified design. Supports only single or concatenated runs.

Inputs:

```
[Mandatory]
TR: (a float)
session_info: (a list of from 1 to 1 items which are any value)
    Session specific information generated by ``modelgen.SpecifyModel``,
    FitGLM does not support multiple runs unless they are concatenated
    (see SpecifyModel options)

[Optional]
drift_model: ('Cosine' or 'Polynomial' or 'Blank', nipyne default
    value: Cosine)
    string that specifies the desired drift model, to be chosen among
    'Polynomial', 'Cosine', 'Blank'
hrf_model: ('Canonical' or 'Canonical With Derivative' or 'FIR',
    nipyne default value: Canonical)
    that specifies the hemodynamic reponse function it can be
    'Canonical', 'Canonical With Derivative' or 'FIR'
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask: (a file name)
    restrict the fitting only to the region defined by this mask
method: ('kalman' or 'ols', nipyne default value: kalman)
    method to fit the model, ols or kalma; kalman is more time consuming
    but it supports autoregressive model
model: ('ar1' or 'spherical', nipyne default value: ar1)
    autoregressive mode is available only for the kalman method
normalize_design_matrix: (a boolean, nipyne default value: False)
    normalize (zscore) the regressors before fitting
plot_design_matrix: (a boolean, nipyne default value: False)
save_residuals: (a boolean, nipyne default value: False)
```

Outputs:

```
a: (an existing file name)
axis: (any value)
beta: (an existing file name)
constants: (any value)
dof: (any value)
nvbeta: (any value)
reg_names: (a list of items which are any value)
```

```
residuals: (a file name)
s2: (an existing file name)
```

interfaces.nipy.preprocess

ComputeMask

[Link to code](#)

Inputs:

```
[Mandatory]
mean_volume: (an existing file name)
    mean EPI image, used to compute the threshold for the mask

[Optional]
M: (a float)
    upper fraction of the histogram to be discarded
cc: (a boolean)
    Keep only the largest connected component
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
m: (a float)
    lower fraction of the histogram to be discarded
reference_volume: (an existing file name)
    reference volume used to compute the mask. If none is give, the mean
    volume is used.
```

Outputs:

```
brain_mask: (an existing file name)
```

FmriRealign4d

[Link to code](#)

Simultaneous motion and slice timing correction algorithm

This interface wraps nipy's FmriRealign4d algorithm¹.

Examples

```
>>> from nipy.interfaces.nipy.preprocess import FmriRealign4d
>>> realigner = FmriRealign4d()
>>> realigner.inputs.in_file = ['functional.nii']
>>> realigner.inputs.tr = 2
>>> realigner.inputs.slice_order = list(range(0,67))
>>> res = realigner.run()
```

References

Inputs:

¹ Roche A. A four-dimensional registration algorithm with application to joint correction of motion and slice timing in fMRI. IEEE Trans Med Imaging. 2011 Aug;30(8):1546-54. DOI.

```
[Mandatory]
in_file: (a list of items which are an existing file name)
        File to realign
tr: (a float)
    TR in seconds

[Optional]
between_loops: (a list of items which are an integer (int or long),
               nipyype default value: [5])
               loops used to realign different runs
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
loops: (a list of items which are an integer (int or long), nipyype
        default value: [5])
        loops within each run
slice_order: (a list of items which are an integer (int or long))
              0 based slice order. This would be equivalent to
              entering np.argsort(spm_slice_order) for this field. This
              effects interleaved acquisition. This field will be deprecated
              in future Nipy releases and be replaced by actual slice acquisition
              times.
              requires: time_interp
speedup: (a list of items which are an integer (int or long), nipyype
          default value: [5])
          successive image sub-sampling factors for acceleration
start: (a float, nipyype default value: 0.0)
        time offset into TR to align slices to
time_interp: (True)
              Assume smooth changes across time e.g., fmri series. If you don't
              want slice timing correction set this to undefined
              requires: slice_order
tr_slices: (a float)
            TR slices
            requires: time_interp
```

Outputs:

```
out_file: (a list of items which are an existing file name)
          Realigned files
par_file: (a list of items which are an existing file name)
          Motion parameter files
```

SpaceTimeRealigner

[Link to code](#)

Simultaneous motion and slice timing correction algorithm

If slice_times is not specified, this algorithm performs spatial motion correction

This interface wraps nipy's SpaceTimeRealign algorithm [\[Roche2011\]](#) or simply the SpatialRealign algorithm when timing info is not provided.

Examples

```
>>> from nipyype.interfaces.nipy import SpaceTimeRealigner
>>> #Run spatial realignment only
>>> realigner = SpaceTimeRealigner()
```

```
>>> realigner.inputs.in_file = ['functional.nii']
>>> res = realigner.run()
```

```
>>> realigner = SpaceTimeRealigner()
>>> realigner.inputs.in_file = ['functional.nii']
>>> realigner.inputs.tr = 2
>>> realigner.inputs.slice_times = list(range(0, 3, 67))
>>> realigner.inputs.slice_info = 2
>>> res = realigner.run()
```

References

Inputs:

[Mandatory]

in_file: (a list of items which are an existing file name)
File to realign

[Optional]

ignore_exception: (a boolean, nipy default value: False)
Print an error message instead of throwing an exception in case the interface fails to run

slice_info: (an integer (int or long) or a list of items which are any value)
Single integer or length 2 sequence If int, the axis in `images` that is the slice axis. In a 4D image, this will often be axis = 2. If a 2 sequence, then elements are `(slice_axis, slice_direction)`, where `slice_axis` is the slice axis in the image as above, and `slice_direction` is 1 if the slices were acquired slice 0 first, slice -1 last, or -1 if acquired slice -1 first, slice 0 last. If `slice_info` is an int, assume `slice_direction` == 1.
requires: slice_times

slice_times: (a list of items which are a float or 'asc_alt_2' or 'asc_alt_2_1' or 'asc_alt_half' or 'asc_alt_siemens' or 'ascending' or 'desc_alt_2' or 'desc_alt_half' or 'descending')
Actual slice acquisition times.

tr: (a float)
TR in seconds
requires: slice_times

Outputs:

out_file: (a list of items which are an existing file name)
Realigned files

par_file: (a list of items which are an existing file name)
Motion parameter files. Angles are **not** euler angles

Trim

[Link to code](#)

Simple interface to trim a few volumes from a 4d fmri nifti file

Examples

```
>>> from nipyype.interfaces.nipy.preprocess import Trim
>>> trim = Trim()
>>> trim.inputs.in_file = 'functional.nii'
>>> trim.inputs.begin_index = 3 # remove 3 first volumes
>>> res = trim.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        EPI image to trim

[Optional]
begin_index: (an integer (int or long), nipyype default value: 0)
            first volume
end_index: (an integer (int or long), nipyype default value: 0)
           last volume indexed as in python (and 0 for last)
ignore_exception: (a boolean, nipyype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
out_file: (a file name)
          output filename
suffix: (a unicode string, nipyype default value: _trim)
        suffix for out_file to use if no out_file provided
```

Outputs:

```
out_file: (an existing file name)
```

interfaces.nipy.utils

Similarity

[Link to code](#)

Calculates similarity between two 3D volumes. Both volumes have to be in the same coordinate system, same space within that coordinate system and with the same voxel dimensions.

Deprecated since version 0.10.0: Use `nipyype.algorithms.metrics.Similarity` instead.

Example

```
>>> from nipyype.interfaces.nipy.utils import Similarity
>>> similarity = Similarity()
>>> similarity.inputs.volume1 = 'rcls1.nii'
>>> similarity.inputs.volume2 = 'rcls2.nii'
>>> similarity.inputs.mask1 = 'mask.nii'
>>> similarity.inputs.mask2 = 'mask.nii'
>>> similarity.inputs.metric = 'cr'
>>> res = similarity.run()
```

Inputs:

```
[Mandatory]
volume1: (an existing file name)
        3D volume
volume2: (an existing file name)
```

```

    3D volume

[Optional]
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
mask1: (an existing file name)
    3D volume
mask2: (an existing file name)
    3D volume
metric: ('cc' or 'cr' or 'crl1' or 'mi' or 'nmi' or 'slr' or a
    callable value, nipyte default value: None)
    str or callable
    Cost-function for assessing image similarity. If a string,
    one of 'cc': correlation coefficient, 'cr': correlation
    ratio, 'crl1': L1-norm based correlation ratio, 'mi': mutual
    information, 'nmi': normalized mutual information, 'slr':
    supervised log-likelihood ratio. If a callable, it should
    take a two-dimensional array representing the image joint
    histogram as an input and return a float.

```

Outputs:

```

similarity: (a float)
    Similarity between volume 1 and 2

```


interfaces.nitime.analysis

CoherenceAnalyzer

[Link to code](#)

Inputs:

```
[Mandatory]

[Optional]
NFFT: (a long integer >= 32, nipy default value: 64)
    This is the size of the window used for the spectral estimation. Use
    values between 32 and the number of samples in your time-
    series. (Defaults to 64.)
TR: (a float)
    The TR used to collect the data in your csv file <in_file>
figure_type: ('matrix' or 'network', nipy default value: matrix)
    The type of plot to generate, where 'matrix' denotes a matrix image
    and 'network' denotes a graph representation. Default: 'matrix'
frequency_range: (a list of from 2 to 2 items which are any value,
    nipy default value: [0.02, 0.15])
    The range of frequencies over which the analysis will
    average. [low, high] (Default [0.02, 0.15])
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_TS: (any value)
    a nitime TimeSeries object
in_file: (an existing file name)
    csv file with ROIs on the columns and time-points on the rows. ROI
    names at the top row
    requires: TR
n_overlap: (a long integer >= 0, nipy default value: 0)
    The number of samples which overlap between subsequent
    windows. (Defaults to 0)
output_csv_file: (a file name)
    File to write outputs (coherence, time-delay) with file-names:
```

```
    file_name_ {coherence,timedelay}
output_figure_file: (a file name)
    File to write output figures (coherence,time-delay) with file-names:
    file_name_{coherence,timedelay}. Possible formats:
    .png,.svg,.pdf,.jpg,...
```

Outputs:

```
coherence_array: (an array)
    The pairwise coherence valuesbetween the ROIs
coherence_csv: (a file name)
    A csv file containing the pairwise coherence values
coherence_fig: (a file name)
    Figure representing coherence values
timedelay_array: (an array)
    The pairwise time delays between theROIs (in seconds)
timedelay_csv: (a file name)
    A csv file containing the pairwise time delay values
timedelay_fig: (a file name)
    Figure representing coherence values
```

interfaces.semtools.brains.classify

BRAINSPosteriorToContinuousClass

[Link to code](#)

Wraps command **BRAINSPosteriorToContinuousClass**

title: Tissue Classification

category: BRAINS.Classify

description: This program will generate an 8-bit continuous tissue classified image based on BRAINSABC posterior images.

version: 3.0

documentation-url: <http://www.nitrc.org/plugins/mwiki/index.php/brains:BRAINSClassify>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Vincent A. Magnotta

acknowledgements: Funding for this work was provided by NIH/NINDS award NS050568

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputBasalGmVolume: (an existing file name)
    Basal Grey Matter Posterior Volume
    flag: --inputBasalGmVolume %s
inputCrblGmVolume: (an existing file name)
    Cerebellum Grey Matter Posterior Volume
    flag: --inputCrblGmVolume %s

```

```
inputCrblWmVolume: (an existing file name)
    Cerebellum White Matter Posterior Volume
    flag: --inputCrblWmVolume %s
inputCsfVolume: (an existing file name)
    CSF Posterior Volume
    flag: --inputCsfVolume %s
inputSurfaceGmVolume: (an existing file name)
    Surface Grey Matter Posterior Volume
    flag: --inputSurfaceGmVolume %s
inputVbVolume: (an existing file name)
    Venous Blood Posterior Volume
    flag: --inputVbVolume %s
inputWhiteVolume: (an existing file name)
    White Matter Posterior Volume
    flag: --inputWhiteVolume %s
outputVolume: (a boolean or a file name)
    Output Continuous Tissue Classified Image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Output Continuous Tissue Classified Image
```

interfaces.semtools.brains.segmentation

BRAINSTalairach

[Link to code](#)

Wraps command **** BRAINSTalairach ****

title: BRAINS Talairach

category: BRAINS.Segmentation

description: This program creates a VTK structured grid defining the Talairach coordinate system based on four points: AC, PC, IRP, and SLA. The resulting structured grid can be written as either a classic VTK file or the new VTK XML file format. Two representations of the resulting grid can be written. The first is a bounding box representation that also contains the location of the AC and PC points. The second representation is the full Talairach grid representation that includes the additional rows of boxes added to the inferior allowing full coverage of the cerebellum.

version: 0.1

documentation-url: <http://www.nitrc.org/plugins/mwiki/index.php/brains:BRAINSTalairach>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Steven Dunn and Vincent Magnotta

acknowledgements: Funding for this work was provided by NIH/NINDS award NS050568

Inputs:

```
[Mandatory]

[Optional]
AC: (a list of items which are a float)
    Location of AC Point
    flag: --AC %s
ACisIndex: (a boolean)
```

```

        AC Point is Index
        flag: --ACisIndex
    IRP: (a list of items which are a float)
        Location of IRP Point
        flag: --IRP %s
    IRPisIndex: (a boolean)
        IRP Point is Index
        flag: --IRPisIndex
    PC: (a list of items which are a float)
        Location of PC Point
        flag: --PC %s
    PCisIndex: (a boolean)
        PC Point is Index
        flag: --PCisIndex
    SLA: (a list of items which are a float)
        Location of SLA Point
        flag: --SLA %s
    SLAisIndex: (a boolean)
        SLA Point is Index
        flag: --SLAisIndex
    args: (a unicode string)
        Additional parameters to the command
        flag: %s
    environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
    ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
    inputVolume: (an existing file name)
        Input image used to define physical space of images
        flag: --inputVolume %s
    outputBox: (a boolean or a file name)
        Name of the resulting Talairach Bounding Box file
        flag: --outputBox %s
    outputGrid: (a boolean or a file name)
        Name of the resulting Talairach Grid file
        flag: --outputGrid %s
    terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputBox: (an existing file name)
    Name of the resulting Talairach Bounding Box file
outputGrid: (an existing file name)
    Name of the resulting Talairach Grid file

```

BRAINSTalairachMask[Link to code](#)Wraps command **** BRAINSTalairachMask ****

title: Talairach Mask

category: BRAINS.Segmentation

description: This program creates a binary image representing the specified Talairach region. The input is an

example image to define the physical space for the resulting image, the Talairach grid representation in VTK format, and the file containing the Talairach box definitions to be generated. These can be combined in BRAINS to create a label map using the procedure `Brains::WorkupUtils::CreateLabelMapFromBinaryImages`.

version: 0.1

documentation-url: <http://www.nitrc.org/plugins/mwiki/index.php/brains:BRAINSTalairachMask>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Steven Dunn and Vincent Magnotta

acknowledgements: Funding for this work was provided by NIH/NINDS award NS050568

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
expand: (a boolean)
    Expand exterior box to include surface CSF
    flag: --expand
hemisphereMode: ('left' or 'right' or 'both')
    Mode for box creation: left, right, both
    flag: --hemisphereMode %s
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input image used to define physical space of resulting mask
    flag: --inputVolume %s
outputVolume: (a boolean or a file name)
    Output filename for the resulting binary image
    flag: --outputVolume %s
talairachBox: (an existing file name)
    Name of the Talairach box file.
    flag: --talairachBox %s
talairachParameters: (an existing file name)
    Name of the Talairach parameter file.
    flag: --talairachParameters %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Output filename for the resulting binary image
```

SimilarityIndex

[Link to code](#)

Wraps command `** SimilarityIndex **`

title: BRAINSCut:SimilarityIndexComputation

category: BRAINS.Segmentation

description: Automatic analysis of BRAINSCut Output

version: 1.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Eunyoun  Regin Kim

Inputs:

```
[Mandatory]

[Optional]
ANNContinuousVolume: (an existing file name)
    ANN Continuous volume to be compared to the manual volume
    flag: --ANNContinuousVolume %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputManualVolume: (an existing file name)
    input manual(reference) volume
    flag: --inputManualVolume %s
outputCSVFilename: (an existing file name)
    output CSV Filename
    flag: --outputCSVFilename %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thresholdInterval: (a float)
    Threshold interval to compute similarity index between zero and one
    flag: --thresholdInterval %f
```

Outputs:

None

interfaces.semtools.brains.utilities

GenerateEdgeMapImage

[Link to code](#)

Wraps command **** GenerateEdgeMapImage ****

title: GenerateEdgeMapImage

category: BRAINS.Utilities

description: Automatic edgemap generation for edge-guided super-resolution reconstruction

version: 1.0

contributor: Ali Ghayoor

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
```

```

    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputMRVolumes: (a list of items which are an existing file name)
    List of input structural MR volumes to create the maximum edgemap
    flag: --inputMRVolumes %s...
inputMask: (an existing file name)
    Input mask file name. If set, image histogram percentiles will be
    calculated within the mask
    flag: --inputMask %s
lowerPercentileMatching: (a float)
    Map lower quantile and below to minOutputRange. It should be a value
    between zero and one
    flag: --lowerPercentileMatching %f
maximumOutputRange: (an integer (int or long))
    Map upper quantile and above to maximum output range. Default is 255
    that is the maximum range of unsigned char
    flag: --maximumOutputRange %d
minimumOutputRange: (an integer (int or long))
    Map lower quantile and below to minimum output range. It should be a
    small number greater than zero. Default is 1
    flag: --minimumOutputRange %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputEdgeMap: (a boolean or a file name)
    output edgemap file name
    flag: --outputEdgeMap %s
outputMaximumGradientImage: (a boolean or a file name)
    output gradient image file name
    flag: --outputMaximumGradientImage %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
upperPercentileMatching: (a float)
    Map upper quantile and above to maxOutputRange. It should be a value
    between zero and one
    flag: --upperPercentileMatching %f

```

Outputs:

```

outputEdgeMap: (an existing file name)
    (required) output file name
outputMaximumGradientImage: (an existing file name)
    output gradient image file name

```

GeneratePurePlugMask[Link to code](#)Wraps command **** GeneratePurePlugMask ****

title: GeneratePurePlugMask

category: BRAINS.Utilities

description: This program gets several modality image files and returns a binary mask that defines the pure plugs

version: 1.0

contributor: Ali Ghayoor

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputImageModalities: (a list of items which are an existing file
    name)
    List of input image file names to create pure plugs mask
    flag: --inputImageModalities %s...
numberOfSubSamples: (a list of items which are an integer (int or
    long))
    Number of continous index samples taken at each direction of lattice
    space for each plug volume
    flag: --numberOfSubSamples %s
outputMaskFile: (a boolean or a file name)
    Output binary mask file name
    flag: --outputMaskFile %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (a float)
    threshold value to define class membership
    flag: --threshold %f
```

Outputs:

```
outputMaskFile: (an existing file name)
    (required) Output binary mask file name
```

HistogramMatchingFilter

[Link to code](#)

Wraps command **** HistogramMatchingFilter ****

title: Write Out Image Intensities

category: BRAINS.Utilities

description: For Analysis

version: 0.1

contributor: University of Iowa Department of Psychiatry, <http://www.psychiatry.uiowa.edu>

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
```

```
Additional parameters to the command
flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
histogramAlgorithm: ('OtsuHistogramMatching')
                  histogram algrithm selection
                  flag: --histogramAlgorithm %s
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
inputBinaryVolume: (an existing file name)
                  inputBinaryVolume
                  flag: --inputBinaryVolume %s
inputVolume: (an existing file name)
              The Input image to be computed for statistics
              flag: --inputVolume %s
numberOfHistogramBins: (an integer (int or long))
                      number of histogram bin
                      flag: --numberOfHistogramBins %d
numberOfMatchPoints: (an integer (int or long))
                    number of histogram matching points
                    flag: --numberOfMatchPoints %d
outputVolume: (a boolean or a file name)
              Output Image File Name
              flag: --outputVolume %s
referenceBinaryVolume: (an existing file name)
                     referenceBinaryVolume
                     flag: --referenceBinaryVolume %s
referenceVolume: (an existing file name)
                 The Input image to be computed for statistics
                 flag: --referenceVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                 Control terminal output: `stream` - displays to terminal immediately
                 (default), `allatonce` - waits till command is finished to display
                 output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
         verbose mode running for debbuging
         flag: --verbose
writeHistogram: (a unicode string)
               decide if histogram data would be written with prefixe of the file
               name
               flag: --writeHistogram %s
```

Outputs:

```
outputVolume: (an existing file name)
              Output Image File Name
```

interfaces.semtools.converters

DWICompare

[Link to code](#)

Wraps command **** DWICompare ****

title: Nrrd DWI comparison

category: Converters

description: Compares two nrrd format DWI images and verifies that gradient magnitudes, gradient directions, measurement frame, and max B0 value are iden­tic­le. Used for testing DWIConvert.

version: 0.1.0.\$Revision: 916 \$(alpha)

documenta­tion-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/DWIConvert>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Mark Scully (UIowa)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Additional support for DTI data produced on Philips scanners was contributed by Vincent Magnotta and Hans Johnson at the University of Iowa.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume1: (an existing file name)
    First input volume (.nhdr or .nrrd)
    flag: --inputVolume1 %s
inputVolume2: (an existing file name)
    Second input volume (.nhdr or .nrrd)
    flag: --inputVolume2 %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

None

DWISimpleCompare

[Link to code](#)

Wraps command **DWISimpleCompare**

title: Nrrd DWI comparison

category: Converters

description: Compares two nrrd format DWI images and verifies that gradient magnitudes, gradient directions, measurement frame, and max B0 value are iden­tic­le. Used for testing DWIConvert.

version: 0.1.0.\$Revision: 916 \$(alpha)

documenta­tion-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/DWIConvert>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Mark Scully (UIowa)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Additional support for DTI data produced on Philips scanners was contributed by Vincent Magnotta and Hans Johnson at the University of Iowa.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
checkDWIData: (a boolean)
    check for existence of DWI data, and if present, compare it
    flag: --checkDWIData
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume1: (an existing file name)
    First input volume (.nhdr or .nrrd)
    flag: --inputVolume1 %s
inputVolume2: (an existing file name)
    Second input volume (.nhdr or .nrrd)
    flag: --inputVolume2 %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

None

interfaces.semtools.diffusion.diffusion

DWIConvert

[Link to code](#)

Wraps command **** DWIConvert ****

title: DWIConverter

category: Diffusion.Diffusion Data Conversion

description: Converts diffusion weighted MR images in dicom series into Nrrd format for analysis in Slicer. This program has been tested on only a limited subset of DTI dicom formats available from Siemens, GE, and Phillips scanners. Work in progress to support dicom multi-frame data. The program parses dicom header to extract necessary information about measurement frame, diffusion weighting directions, b-values, etc, and write out a nrrd image. For non-diffusion weighted dicom images, it loads in an entire dicom series and writes out a single dicom volume in a .nhdr/.raw pair.

version: Version 1.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/DWIConverter>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Vince Magnotta (UIowa), Hans Johnson (UIowa), Joy Matsui (UIowa), Kent Williams (UIowa), Mark Scully (Uiowa), Xiaodong Tao (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Additional support for DTI data produced on Philips scanners was contributed by Vincent Magnotta and Hans Johnson at the University of Iowa.

Inputs:

```

[Mandatory]

[Optional]
allowLossyConversion: (a boolean)
    The only supported output type is 'short'. Conversion from images of
    a different type may cause data loss due to rounding or truncation.
    Use with caution!
    flag: --allowLossyConversion
args: (a unicode string)
    Additional parameters to the command
    flag: %s
conversionMode: ('DicomToNrrd' or 'DicomToFSL' or 'NrrdToFSL' or
    'FSLToNrrd')
    Determine which conversion to perform. DicomToNrrd (default):
    Convert DICOM series to NRRD DicomToFSL: Convert DICOM series to
    NIfTI File + gradient/bvalue text files NrrdToFSL: Convert DWI NRRD
    file to NIfTI File + gradient/bvalue text files FSLToNrrd: Convert
    NIfTI File + gradient/bvalue text files to NRRD file.
    flag: --conversionMode %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fMRI: (a boolean)
    Output a NRRD file, but without gradients
    flag: --fMRI
fslNIFTIFile: (an existing file name)
    4D NIfTI file containing gradient volumes
    flag: --fslNIFTIFile %s
gradientVectorFile: (a boolean or a file name)
    Text file giving gradient vectors
    flag: --gradientVectorFile %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputBValues: (an existing file name)
    The B Values are stored in FSL .bval text file format
    flag: --inputBValues %s
inputBVectors: (an existing file name)
    The Gradient Vectors are stored in FSL .bvec text file format
    flag: --inputBVectors %s
inputDicomDirectory: (an existing directory name)
    Directory holding Dicom series
    flag: --inputDicomDirectory %s
inputVolume: (an existing file name)
    Input DWI volume -- not used for DicomToNrrd mode.
    flag: --inputVolume %s
outputBValues: (a boolean or a file name)
    The B Values are stored in FSL .bval text file format (defaults to
    <outputVolume>.bval)
    flag: --outputBValues %s
outputBVectors: (a boolean or a file name)
    The Gradient Vectors are stored in FSL .bvec text file format
    (defaults to <outputVolume>.bvec)
    flag: --outputBVectors %s
outputDirectory: (a boolean or a directory name)
    Directory holding the output NRRD file

```

```

    flag: --outputDirectory %s
outputVolume: (a boolean or a file name)
    Output filename (.nhdr or .nrrd)
    flag: --outputVolume %s
smallGradientThreshold: (a float)
    If a gradient magnitude is greater than 0 and less than
    smallGradientThreshold, then DWIConvert will display an error
    message and quit, unless the useBMatrixGradientDirections option is
    set.
    flag: --smallGradientThreshold %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transposeInputBVectors: (a boolean)
    FSL input BVectors are expected to be encoded in the input file as
    one vector per line. If it is not the case, use this option to
    transpose the file as it is read.
    flag: --transposeInputBVectors
useBMatrixGradientDirections: (a boolean)
    Fill the nhdr header with the gradient directions and bvalues
    computed out of the BMatrix. Only changes behavior for Siemens data.
    In some cases the standard public gradients are not properly
    computed. The gradients can emperically computed from the private
    BMatrix fields. In some cases the private BMatrix is consistent with
    the public grandients, but not in all cases, when it exists BMatrix
    is usually most robust.
    flag: --useBMatrixGradientDirections
useIdentityMeaseurementFrame: (a boolean)
    Adjust all the gradients so that the measurement frame is an
    identity matrix.
    flag: --useIdentityMeaseurementFrame
writeProtocolGradientsFile: (a boolean)
    Write the protocol gradients to a file suffixed by '.txt' as they
    were specified in the procol by multiplying each diffusion gradient
    direction by the measurement frame. This file is for debugging
    purposes only, the format is not fixed, and will likely change as
    debugging of new dicom formats is necessary.
    flag: --writeProtocolGradientsFile

```

Outputs:

```

gradientVectorFile: (an existing file name)
    Text file giving gradient vectors
outputBValues: (an existing file name)
    The B Values are stored in FSL .bval text file format (defaults to
    <outputVolume>.bval)
outputBVectors: (an existing file name)
    The Gradient Vectors are stored in FSL .bvec text file format
    (defaults to <outputVolume>.bvec)
outputDirectory: (an existing directory name)
    Directory holding the output NRRD file
outputVolume: (an existing file name)
    Output filename (.nhdr or .nrrd)

```

dtiaverage[Link to code](#)

Wraps command `** dtiaverage **`

title: DTIAverage (DTIProcess)

category: Diffusion.Diffusion Tensor Images.CommandLineOnly

description: `dtiaverage` is a program that allows to compute the average of an arbitrary number of tensor fields (listed after)

Several average method can be used (specified by the `--method` option): euclidian, log-euclidian and pga.

The default being euclidian.

version: 1.0.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Extensions/DTIProcess>

license: Copyright (c) Casey Goodlett. All rights reserved. See <http://www.ia.unc.edu/dev/Copyright.htm> for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett

Inputs:

```
[Mandatory]

[Optional]
DTI_double: (a boolean)
    Tensor components are saved as doubles (cannot be visualized in
    Slicer)
    flag: --DTI_double
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputs: (a list of items which are an existing file name)
    List of all the tensor fields to be averaged
    flag: --inputs %s...
tensor_output: (a boolean or a file name)
    Averaged tensor volume
    flag: --tensor_output %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    produce verbose output
    flag: --verbose
```

Outputs:

```
tensor_output: (an existing file name)
    Averaged tensor volume
```

dtiestim

[Link to code](#)

Wraps command `** dtiestim **`

title: DTIEstim (DTIProcess)

category: Diffusion.Diffusion Weighted Images

description: dtiestim is a tool that takes in a set of DWIs (with `-dwi_image` option) in nrrd format and estimates a tensor field out of it. The output tensor file name is specified with the `-tensor_output` option. There are several methods to estimate the tensors which you can specify with the option `-method` `llslwlsnlslml`. Here is a short description of the different methods:

- lls** Linear least squares. Standard estimation technique that recovers the tensor parameters by multiplying the log of the normalized signal intensities by the pseudo-inverse of the gradient matrix. Default option.
- wls** Weighted least squares. This method is similar to the linear least squares method except that the gradient matrix is weighted by the original lls estimate. (See Salvador, R., Pena, A., Menon, D. K., Carpenter, T. A., Pickard, J. D., and Bullmore, E. T. Formal characterization and extension of the linearized diffusion tensor model. Human Brain Mapping 24, 2 (Feb. 2005), 144-155. for more information on this method). This method is recommended for most applications. The weight for each iteration can be specified with the `-weight_iterations`. It is not currently the default due to occasional matrix singularities.
- nls** Non-linear least squares. This method does not take the log of the signal and requires an optimization based on levenberg-marquadt to optimize the parameters of the signal. The lls estimate is used as an initialization. For this method the step size can be specified with the `-step` option.
- ml** Maximum likelihood estimation. This method is experimental and is not currently recommended. For this ml method the sigma can be specified with the option `-sigma` and the step size can be specified with the `-step` option.

You can set a threshold (`-threshold`) to have the tensor estimated to only a subset of voxels. All the baseline voxel value higher than the threshold define the voxels where the tensors are computed. If not specified the threshold is calculated using an OTSU threshold on the baseline image. The masked generated by the `-t` option or by the otsu value can be saved with the `-B0_mask_output` option.

dtiestim also can extract a few scalar images out of the DWI set of images:

- the average baseline image (`-B0`) which is the average of all the B0s.
- the IDWI (`-idwi`) which is the geometric mean of the diffusion images.

You can also load a mask if you want to compute the tensors only where the voxels are non-zero (`-brain_mask`) or a negative mask and the tensors will be estimated where the negative mask has zero values (`-bad_region_mask`)
version: 1.2.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Extensions/DTIPProcess>

license: Copyright (c) Casey Goodlett. All rights reserved.

See <http://www.ia.unc.edu/dev/Copyright.htm> for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett, Francois Budin

acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); (1=University of Iowa Department of Psychiatry, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering) provided conversions to make DTIPProcess compatible with Slicer execution, and simplified the stand-alone build requirements by removing the dependancies on boost and a fortran compiler.

Inputs:

```
[Mandatory]

[Optional]
B0: (a boolean or a file name)
    Baseline image, average of all baseline images
    flag: --B0 %s
B0_mask_output: (a boolean or a file name)
    B0 mask used for the estimation. B0 thresholded either with the -t
    option value or the automatic OTSU value
    flag: --B0_mask_output %s
DTI_double: (a boolean)
    Tensor components are saved as doubles (cannot be visualized in
    Slicer)
    flag: --DTI_double
args: (a unicode string)
```



```

        Additional parameters to the command
        flag: %s
bad_region_mask: (an existing file name)
        Bad region mask. Image where for every voxel > 0 the tensors are not
        estimated
        flag: --bad_region_mask %s
brain_mask: (an existing file name)
        Brain mask. Image where for every voxel == 0 the tensors are not
        estimated. Be aware that in addition a threshold based masking will
        be performed by default. If such an additional threshold masking is
        NOT desired, then use option -t 0.
        flag: --brain_mask %s
correction: ('none' or 'zero' or 'abs' or 'nearest')
        Correct the tensors if computed tensor is not semi-definite positive
        flag: --correction %s
defaultTensor: (a list of items which are a float)
        Default tensor used if estimated tensor is below a given threshold
        flag: --defaultTensor %s
dwi_image: (an existing file name)
        DWI image volume (required)
        flag: --dwi_image %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
idwi: (a boolean or a file name)
        idwi output image. Image with isotropic diffusion-weighted
        information = geometric mean of diffusion images
        flag: --idwi %s
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
method: ('lls' or 'wls' or 'nls' or 'ml')
        Estimation method (lls:linear least squares, wls:weighted least
        squares, nls:non-linear least squares, ml:maximum likelihood)
        flag: --method %s
shiftNeg: (a boolean)
        Shift eigenvalues so all are positive (accounts for bad tensors
        related to noise or acquisition error). This is the same option as
        the one available in DWIToDTIEstimation in Slicer (but instead of
        just adding the minimum eigenvalue to all the eigenvalues if it is
        smaller than 0, we use a coefficient to have strictly positive
        eigenvalues
        flag: --shiftNeg
shiftNegCoeff: (a float)
        Shift eigenvalues so all are positive (accounts for bad tensors
        related to noise or acquisition error). Instead of just adding the
        minimum eigenvalue to all the eigenvalues if it is smaller than 0,
        we use a coefficient to have strictly positive eigenvalues.
        Coefficient must be between 1.0 and 1.001 (included).
        flag: --shiftNegCoeff %f
sigma: (a float)
        flag: --sigma %f
step: (a float)
        Gradient descent step size (for nls and ml methods)
        flag: --step %f
tensor_output: (a boolean or a file name)
        Tensor OutputImage

```

```

    flag: --tensor_output %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (an integer (int or long))
    Baseline threshold for estimation. If not specified calculated using
    an OTSU threshold on the baseline image.
    flag: --threshold %d
verbose: (a boolean)
    produce verbose output
    flag: --verbose
weight_iterations: (an integer (int or long))
    Number of iterations to recalculate weightings from tensor estimate
    flag: --weight_iterations %d

```

Outputs:

```

B0: (an existing file name)
    Baseline image, average of all baseline images
B0_mask_output: (an existing file name)
    B0 mask used for the estimation. B0 thresholded either with the -t
    option value or the automatic OTSU value
idwi: (an existing file name)
    idwi output image. Image with isotropic diffusion-weighted
    information = geometric mean of diffusion images
tensor_output: (an existing file name)
    Tensor OutputImage

```

dtiprocess

Link to code

Wraps command `** dtiprocess **`

title: DTIPProcess (DTIPProcess)

category: Diffusion.Diffusion Tensor Images

description: dtiprocess is a tool that handles tensor fields. It takes as an input a tensor field in nrrd format. It can generate diffusion scalar properties out of the tensor field such as : FA (`-fa_output`), Gradient FA image (`-fa_gradient_output`), color FA (`-color_fa_output`), MD (`-md_output`), Frobenius norm (`-frobenius_norm_output`), lbd1, lbd2, lbd3 (`-lambda{1,2,3}_output`), binary map of voxel where if any of the eigenvalue is negative, the voxel is set to 1 (`-negative_eigenvector_output`)

It also creates 4D images out of the tensor field such as: Highest eigenvector map (highest eigenvector at each voxel) (`-principal_eigenvector_output`)

Masking capabilities: For any of the processing done with dtiprocess, it's possible to apply it on a masked region of the tensor field. You need to use the `-mask` option for any of the option to be applied on that tensor field sub-region only. If you want to save the masked tensor field use the option `-outmask` and specify the new masked tensor field file name. dtiprocess also allows a range of transformations on the tensor fields. The transformed tensor field file name is specified with the option `-deformation_output`. There are 3 resampling interpolation methods specified with the tag `-interpolation` followed by the type to use (nearestneighbor, linear, cubic) Then you have several transformations possible to apply:

- Affine transformations using as an input
- itk affine transformation file (based on the `itkAffineTransform` class)
- Affine transformations using rview (details and download at <http://www.doc.ic.ac.uk/~dr/software/>). There are 2 versions of rview both creating transformation files called dof files. The old version of rview outputs text files containing the transformation parameters. It can be read in with the `-dof_file` option. The new version outputs binary dof files. These dof files can be transformed into human readable file with the dof2mat tool which is part of the rview package. So you need to save the output of dof2mat into a text file which can then be used with the

– newdof_file option. Usage example: `dof2mat mynewdoffile.dof >> mynewdoffile.txt dtiprocess -dti_image mytensorfield.nhdr -newdof_file mynewdoffile.txt -rot_output myaffinetensorfield.nhdr`

Non linear transformations as an input: The default transformation file type is d-field (displacement field) in nrrd format. The option to use is –forward with the name of the file. If the transformation file is a h-field you have to add the option –hField.

version: 1.0.1

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Extensions/DTIProcess>

license: Copyright (c) Casey Goodlett. All rights reserved.

See <http://www.ia.unc.edu/dev/Copyright.htm> for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett

Inputs:

```
[Mandatory]

[Optional]
DTI_double: (a boolean)
    Tensor components are saved as doubles (cannot be visualized in
    Slicer)
    flag: --DTI_double
RD_output: (a boolean or a file name)
    RD (Radial Diffusivity  $1/2 * (\lambda_2 + \lambda_3)$ ) output
    flag: --RD_output %s
affineitk_file: (an existing file name)
    Transformation file for affine transformation. ITK format.
    flag: --affineitk_file %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
color_fa_output: (a boolean or a file name)
    Color Fractional Anisotropy output file
    flag: --color_fa_output %s
correction: ('none' or 'zero' or 'abs' or 'nearest')
    Correct the tensors if computed tensor is not semi-definite positive
    flag: --correction %s
deformation_output: (a boolean or a file name)
    Warped tensor field based on a deformation field. This option
    requires the --forward, -F transformation to be specified.
    flag: --deformation_output %s
dof_file: (an existing file name)
    Transformation file for affine transformation. This can be ITK
    format (or the outdated RView).
    flag: --dof_file %s
dti_image: (an existing file name)
    DTI tensor volume
    flag: --dti_image %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fa_gradient_output: (a boolean or a file name)
    Fractional Anisotropy Gradient output file
    flag: --fa_gradient_output %s
fa_gradmag_output: (a boolean or a file name)
    Fractional Anisotropy Gradient Magnitude output file
    flag: --fa_gradmag_output %s
fa_output: (a boolean or a file name)
```

```
Fractional Anisotropy output file
flag: --fa_output %s
forward: (an existing file name)
    Forward transformation. Assumed to be a deformation field in world
    coordinates, unless the --h-field option is specified.
    flag: --forward %s
frobenius_norm_output: (a boolean or a file name)
    Frobenius Norm Output
    flag: --frobenius_norm_output %s
hField: (a boolean)
    forward and inverse transformations are h-fields instead of
    displacement fields
    flag: --hField
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interpolation: ('nearestneighbor' or 'linear' or 'cubic')
    Interpolation type (nearestneighbor, linear, cubic)
    flag: --interpolation %s
lambda1_output: (a boolean or a file name)
    Axial Diffusivity - Lambda 1 (largest eigenvalue) output
    flag: --lambda1_output %s
lambda2_output: (a boolean or a file name)
    Lambda 2 (middle eigenvalue) output
    flag: --lambda2_output %s
lambda3_output: (a boolean or a file name)
    Lambda 3 (smallest eigenvalue) output
    flag: --lambda3_output %s
mask: (an existing file name)
    Mask tensors. Specify --outmask if you want to save the masked
    tensor field, otherwise the mask is applied just for the current
    processing
    flag: --mask %s
md_output: (a boolean or a file name)
    Mean Diffusivity output file
    flag: --md_output %s
negative_eigenvector_output: (a boolean or a file name)
    Negative Eigenvectors Output: create a binary image where if any of
    the eigen value is below zero, the voxel is set to 1, otherwise 0.
    flag: --negative_eigenvector_output %s
newdof_file: (an existing file name)
    Transformation file for affine transformation. RView NEW format.
    (txt file output of dof2mat)
    flag: --newdof_file %s
outmask: (a boolean or a file name)
    Name of the masked tensor field.
    flag: --outmask %s
principal_eigenvector_output: (a boolean or a file name)
    Principal Eigenvectors Output
    flag: --principal_eigenvector_output %s
reorientation: ('fs' or 'ppd')
    Reorientation type (fs, ppd)
    flag: --reorientation %s
rot_output: (a boolean or a file name)
    Rotated tensor output file. Must also specify the dof file.
    flag: --rot_output %s
scalar_float: (a boolean)
    Write scalar [FA,MD] as unscaled float (with their actual values,
```

```

        otherwise scaled by 10 000). Also causes FA to be unscaled [0..1].
        flag: --scalar_float
sigma: (a float)
        Scale of gradients
        flag: --sigma %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
        produce verbose output
        flag: --verbose

```

Outputs:

```

RD_output: (an existing file name)
        RD (Radial Diffusivity  $1/2 * (\lambda_2 + \lambda_3)$ ) output
color_fa_output: (an existing file name)
        Color Fractional Anisotropy output file
deformation_output: (an existing file name)
        Warped tensor field based on a deformation field. This option
        requires the --forward, -F transformation to be specified.
fa_gradient_output: (an existing file name)
        Fractional Anisotropy Gradient output file
fa_gradmag_output: (an existing file name)
        Fractional Anisotropy Gradient Magnitude output file
fa_output: (an existing file name)
        Fractional Anisotropy output file
frobenius_norm_output: (an existing file name)
        Frobenius Norm Output
lambda1_output: (an existing file name)
        Axial Diffusivity - Lambda 1 (largest eigenvalue) output
lambda2_output: (an existing file name)
        Lambda 2 (middle eigenvalue) output
lambda3_output: (an existing file name)
        Lambda 3 (smallest eigenvalue) output
md_output: (an existing file name)
        Mean Diffusivity output file
negative_eigenvector_output: (an existing file name)
        Negative Eigenvectors Output: create a binary image where if any of
        the eigen value is below zero, the voxel is set to 1, otherwise 0.
outmask: (an existing file name)
        Name of the masked tensor field.
principal_eigenvector_output: (an existing file name)
        Principal Eigenvectors Output
rot_output: (an existing file name)
        Rotated tensor output file. Must also specify the dof file.

```

interfaces.semtools.diffusion.gtract**compareTractInclusion**[Link to code](#)Wraps command **** compareTractInclusion ****

title: Compare Tracts

category: Diffusion.GTRACT

description: This program will halt with a status code indicating whether a test tract is nearly enough included

in a standard tract in the sense that every fiber in the test tract has a low enough sum of squares distance to some fiber in the standard tract modulo spline resampling of every fiber to a fixed number of points.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
closeness: (a float)
    Closeness of every test fiber to some fiber in the standard tract,
    computed as a sum of squares of spatial differences of standard
    points
    flag: --closeness %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
numberOfPoints: (an integer (int or long))
    Number of points in comparison fiber pairs
    flag: --numberOfPoints %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
standardFiber: (an existing file name)
    Required: standard fiber tract file name
    flag: --standardFiber %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
testFiber: (an existing file name)
    Required: test fiber tract file name
    flag: --testFiber %s
testForBijection: (a boolean)
    Flag to apply the closeness criterion both ways
    flag: --testForBijection
testForFiberCardinality: (a boolean)
    Flag to require the same number of fibers in both tracts
    flag: --testForFiberCardinality
writeXMLPolyDataFile: (a boolean)
    Flag to make use of XML files when reading and writing vtkPolyData.
    flag: --writeXMLPolyDataFile
```

Outputs:

None

extractNrrdVectorIndex

[Link to code](#)Wraps command `** extractNrrdVectorIndex **`

title: Extract Nrrd Index

category: Diffusion.GTRACT

description: This program will extract a 3D image (single vector) from a vector 3D image at a given vector index.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Required: input file containing the vector that will be extracted
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the vector image at
    the given index
    flag: --outputVolume %s
setImageOrientation: ('AsAcquired' or 'Axial' or 'Coronal' or
    'Sagittal')
    Sets the image orientation of the extracted vector (Axial, Coronal,
    Sagittal)
    flag: --setImageOrientation %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
vectorIndex: (an integer (int or long))
    Index in the vector image to extract
    flag: --vectorIndex %d
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing the vector image at
    the given index
```

gtractAnisotropyMap

[Link to code](#)

Wraps command `** gtractAnisotropyMap **`

title: Anisotropy Map

category: Diffusion.GTRACT

description: This program will generate a scalar map of anisotropy, given a tensor representation. Anisotropy images are used for fiber tracking, but the anisotropy scalars are not defined along the path. Instead, the tensor representation is included as point data allowing all of these metrics to be computed using only the fiber tract point data. The images can be saved in any ITK supported format, but it is suggested that you use an image format that supports the definition of the image origin. This includes NRRD, NifTI, and Meta formats. These images can also be used for scalar analysis including regional anisotropy measures or VBM style analysis.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
anisotropyType: ('ADC' or 'FA' or 'RA' or 'VR' or 'AD' or 'RD' or
    'LI')
    Anisotropy Mapping Type: ADC, FA, RA, VR, AD, RD, LI
    flag: --anisotropyType %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputTensorVolume: (an existing file name)
    Required: input file containing the diffusion tensor image
    flag: --inputTensorVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the selected kind of
    anisotropy scalar.
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:


```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing the selected kind of
    anisotropy scalar.
```

gtractAverageBvalues

[Link to code](#)

Wraps command `** gtractAverageBvalues **`

title: Average B-Values

category: Diffusion.GTRACT

description: This program will directly average together the baseline gradients (b value equals 0) within a DWI scan. This is usually used after `gtractCoregBvalues`.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
averageB0only: (a boolean)
    Average only baseline gradients. All other gradient directions are
    not averaged, but retained in the outputVolume
    flag: --averageB0only
directionsTolerance: (a float)
    Tolerance for matching identical gradient direction pairs
    flag: --directionsTolerance %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Required: input image file name containing multiple baseline
    gradients to average
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing directly averaged
    baseline images
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing directly averaged
    baseline images
```

gtractClipAnisotropy

[Link to code](#)

Wraps command **** gtractClipAnisotropy ****

title: Clip Anisotropy

category: Diffusion.GTRACT

description: This program will zero the first and/or last slice of an anisotropy image, creating a clipped anisotropy image.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clipFirstSlice: (a boolean)
    Clip the first slice of the anisotropy image
    flag: --clipFirstSlice
clipLastSlice: (a boolean)
    Clip the last slice of the anisotropy image
    flag: --clipLastSlice
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Required: input image file name
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the clipped anisotropy
    image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing the clipped anisotropy
    image
```

gtractCoRegAnatomy

[Link to code](#)

Wraps command **** gtractCoRegAnatomy ****

title: Coregister B0 to Anatomy B-Spline

category: Diffusion.GTRACT

description: This program will register a Nrrd diffusion weighted 4D vector image to a fixed anatomical image. Two registration methods are supported for alignment with anatomical images: Rigid and B-Spline. The rigid registration performs a rigid body registration with the anatomical images and should be done as well to initialize the B-Spline transform. The B-Spline transform is the deformable transform, where the user can control the amount of deformation based on the number of control points as well as the maximum distance that these points can move. The B-Spline registration places a low dimensional grid in the image, which is deformed. This allows for some susceptibility related distortions to be removed from the diffusion weighted images. In general the amount of motion in the slice selection and read-out directions direction should be kept low. The distortion is in the phase encoding direction in the images. It is recommended that skull stripped (i.e. image containing only brain with skull removed) images should be used for image co-registration with the B-Spline transform.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
borderSize: (an integer (int or long))
    Size of border
    flag: --borderSize %d
convergence: (a float)
    Convergence Factor
    flag: --convergence %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gradientTolerance: (a float)
    Gradient Tolerance
    flag: --gradientTolerance %f
gridSize: (a list of items which are an integer (int or long))
    Number of grid subdivisions in all 3 directions
    flag: --gridSize %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputAnatomicalVolume: (an existing file name)
    Required: input anatomical image file name. It is recommended that
    that the input anatomical image has been skull stripped and has the
```

```

    same orientation as the DWI scan.
    flag: --inputAnatomicalVolume %s
inputRigidTransform: (an existing file name)
    Required (for B-Spline type co-registration): input rigid transform
    file name. Used as a starting point for the anatomical B-Spline
    registration.
    flag: --inputRigidTransform %s
inputVolume: (an existing file name)
    Required: input vector image file name. It is recommended that the
    input volume is the skull stripped baseline image of the DWI scan.
    flag: --inputVolume %s
maxBSplineDisplacement: (a float)
    Sets the maximum allowed displacements in image physical
    coordinates for BSpline control grid along each axis. A value of 0.0
    indicates that the problem should be unbounded. NOTE: This only
    constrains the BSpline portion, and does not limit the displacement
    from the associated bulk transform. This can lead to a substantial
    reduction in computation time in the BSpline optimizer.,
    flag: --maxBSplineDisplacement %f
maximumStepSize: (a float)
    Maximum permitted step size to move in the selected 3D fit
    flag: --maximumStepSize %f
minimumStepSize: (a float)
    Minimum required step size to move in the selected 3D fit without
    converging -- decrease this to make the fit more exacting
    flag: --minimumStepSize %f
numberOfHistogramBins: (an integer (int or long))
    Number of histogram bins
    flag: --numberOfHistogramBins %d
numberOfIterations: (an integer (int or long))
    Number of iterations in the selected 3D fit
    flag: --numberOfIterations %d
numberOfSamples: (an integer (int or long))
    The number of voxels sampled for mutual information computation.
    Increase this for a slower, more careful fit. NOTE that it is
    suggested to use samplingPercentage instead of this option. However,
    if set, it overwrites the samplingPercentage option.
    flag: --numberOfSamples %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputTransformName: (a boolean or a file name)
    Required: filename for the fit transform.
    flag: --outputTransformName %s
relaxationFactor: (a float)
    Fraction of gradient from Jacobian to attempt to move in the
    selected 3D fit
    flag: --relaxationFactor %f
samplingPercentage: (a float)
    This is a number in (0.0,1.0] interval that shows the percentage of
    the input fixed image voxels that are sampled for mutual information
    computation. Increase this for a slower, more careful fit. You can
    also limit the sampling focus with ROI masks and ROIAUTO mask
    generation. The default is to use approximately 5% of voxels (for
    backwards compatibility 5% ~= 500000/(256*256*256)). Typical values
    range from 1% for low detail images to 20% for high detail images.
    flag: --samplingPercentage %f
spatialScale: (an integer (int or long))

```

```

        Scales the number of voxels in the image by this value to specify
        the number of voxels used in the registration
        flag: --spatialScale %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
transformType: ('Rigid' or 'Bspline')
        Transform Type: Rigid|Bspline
        flag: --transformType %s
translationScale: (a float)
        How much to scale up changes in position compared to unit rotational
        changes in radians -- decrease this to put more translation in the
        fit
        flag: --translationScale %f
useCenterOfHeadAlign: (a boolean)
        CenterOfHeadAlign attempts to find a hemisphere full of foreground
        voxels from the superior direction as an estimate of where the
        center of a head shape would be to drive a center of mass estimate.
        Perform a CenterOfHeadAlign registration as part of the sequential
        registration steps. This option MUST come first, and CAN NOT be used
        with either MomentsAlign, GeometryAlign, or initialTransform file.
        This family of options superceeds the use of transformType if any of
        them are set.
        flag: --useCenterOfHeadAlign
useGeometryAlign: (a boolean)
        GeometryAlign on assumes that the center of the voxel lattice of the
        images represent similar structures. Perform a GeometryCenterAlign
        registration as part of the sequential registration steps. This
        option MUST come first, and CAN NOT be used with either
        MomentsAlign, CenterOfHeadAlign, or initialTransform file. This
        family of options superceeds the use of transformType if any of them
        are set.
        flag: --useGeometryAlign
useMomentsAlign: (a boolean)
        MomentsAlign assumes that the center of mass of the images represent
        similar structures. Perform a MomentsAlign registration as part of
        the sequential registration steps. This option MUST come first, and
        CAN NOT be used with either CenterOfHeadLAlign, GeometryAlign, or
        initialTransform file. This family of options superceeds the use of
        transformType if any of them are set.
        flag: --useMomentsAlign
vectorIndex: (an integer (int or long))
        Vector image index in the moving image (within the DWI) to be used
        for registration.
        flag: --vectorIndex %d

```

Outputs:

```

outputTransformName: (an existing file name)
        Required: filename for the fit transform.

```

gtractConcatDwi[Link to code](#)Wraps command **** gtractConcatDwi ****

title: Concat DWI Images

category: Diffusion.GTRACT

description: This program will concatenate two DTI runs together.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignoreOrigins: (a boolean)
    If image origins are different force all images to origin of first
    image
    flag: --ignoreOrigins
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (a list of items which are an existing file name)
    Required: input file containing the first diffusion weighted image
    flag: --inputVolume %s...
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the combined diffusion
    weighted images.
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing the combined diffusion
    weighted images.
```

gtractCopyImageOrientation

[Link to code](#)

Wraps command **** gtractCopyImageOrientation ****

title: Copy Image Orientation

category: Diffusion.GTRACT

description: This program will copy the orientation from the reference image into the moving image. Currently, the registration process requires that the diffusion weighted images and the anatomical images have the same image orientation (i.e. Axial, Coronal, Sagittal). It is suggested that you copy the image orientation from the diffusion weighted images and apply this to the anatomical image. This image can be subsequently removed

after the registration step is complete. We anticipate that this limitation will be removed in future versions of the registration programs.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputReferenceVolume: (an existing file name)
    Required: input file containing orientation that will be cloned.
    flag: --inputReferenceVolume %s
inputVolume: (an existing file name)
    Required: input file containing the signed short image to reorient
    without resampling.
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD or Nifti file containing the
    reoriented image in reference image space.
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD or Nifti file containing the
    reoriented image in reference image space.
```

gtractCoregBvalues

[Link to code](#)

Wraps command **** gtractCoregBvalues ****

title: Coregister B-Values

category: Diffusion.GTRACT

description: This step should be performed after converting DWI scans from DICOM to NRRD format. This program will register all gradients in a NRRD diffusion weighted 4D vector image (moving image) to a specified index in a fixed image. It also supports co-registration with a T2 weighted image or field map in the same plane

as the DWI data. The fixed image for the registration should be a b0 image. A mutual information metric cost function is used for the registration because of the differences in signal intensity as a result of the diffusion gradients. The full affine allows the registration procedure to correct for eddy current distortions that may exist in the data. If the eddyCurrentCorrection is enabled, relaxationFactor (0.25) and maximumStepSize (0.1) should be adjusted.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debugLevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
    flag: --debugLevel %d
eddyCurrentCorrection: (a boolean)
    Flag to perform eddy current corection in addition to motion
    correction (recommended)
    flag: --eddyCurrentCorrection
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixedVolume: (an existing file name)
    Required: input fixed image file name. It is recommended that this
    image should either contain or be a b0 image.
    flag: --fixedVolume %s
fixedVolumeIndex: (an integer (int or long))
    Index in the fixed image for registration. It is recommended that
    this image should be a b0 image.
    flag: --fixedVolumeIndex %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
maximumStepSize: (a float)
    Maximum permitted step size to move in each 3D fit step (adjust when
    eddyCurrentCorrection is enabled; suggested value = 0.1)
    flag: --maximumStepSize %f
minimumStepSize: (a float)
    Minimum required step size to move in each 3D fit step without
    converging -- decrease this to make the fit more exacting
    flag: --minimumStepSize %f
movingVolume: (an existing file name)
    Required: input moving image file name. In order to register
    gradients within a scan to its first gradient, set the movingVolume
    and fixedVolume as the same image.
    flag: --movingVolume %s
numberOfIterations: (an integer (int or long))
    Number of iterations in each 3D fit
    flag: --numberOfIterations %d
```



```

numberOfSpatialSamples: (an integer (int or long))
    The number of voxels sampled for mutual information computation.
    Increase this for a slower, more careful fit. NOTE that it is
    suggested to use samplingPercentage instead of this option. However,
    if set, it overwrites the samplingPercentage option.
    flag: --numberOfSpatialSamples %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputTransform: (a boolean or a file name)
    Registration 3D transforms concatenated in a single output file.
    There are no tools that can use this, but can be used for debugging
    purposes.
    flag: --outputTransform %s
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing moving images
    individually resampled and fit to the specified fixed image index.
    flag: --outputVolume %s
registerB0Only: (a boolean)
    Register the B0 images only
    flag: --registerB0Only
relaxationFactor: (a float)
    Fraction of gradient from Jacobian to attempt to move in each 3D fit
    step (adjust when eddyCurrentCorrection is enabled; suggested value
    = 0.25)
    flag: --relaxationFactor %f
samplingPercentage: (a float)
    This is a number in (0.0,1.0] interval that shows the percentage of
    the input fixed image voxels that are sampled for mutual information
    computation. Increase this for a slower, more careful fit. You can
    also limit the sampling focus with ROI masks and ROIAUTO mask
    generation. The default is to use approximately 5% of voxels (for
    backwards compatibility 5% ~= 500000/(256*256*256)). Typical values
    range from 1% for low detail images to 20% for high detail images.
    flag: --samplingPercentage %f
spatialScale: (a float)
    How much to scale up changes in position compared to unit rotational
    changes in radians -- decrease this to put more rotation in the fit
    flag: --spatialScale %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputTransform: (an existing file name)
    Registration 3D transforms concatenated in a single output file.
    There are no tools that can use this, but can be used for debugging
    purposes.
outputVolume: (an existing file name)
    Required: name of output NRRD file containing moving images
    individually resampled and fit to the specified fixed image index.

```

gtractCostFastMarching[Link to code](#)Wraps command **** gtractCostFastMarching ****

title: Cost Fast Marching

category: Diffusion.GTRACT

description: This program will use a fast marching fiber tracking algorithm to identify fiber tracts from a tensor image. This program is the first portion of the algorithm. The user must first run `gtractFastMarchingTracking` to generate the actual fiber tracts. This algorithm is roughly based on the work by G. Parker et al. from IEEE Transactions On Medical Imaging, 21(5): 505-512, 2002. An additional feature of including anisotropy into the `vcl_cost` function calculation is included.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris. The original code here was developed by Daisy Espino.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
anisotropyWeight: (a float)
    Anisotropy weight used for vcl_cost function calculations
    flag: --anisotropyWeight %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputAnisotropyVolume: (an existing file name)
    Required: input anisotropy image file name
    flag: --inputAnisotropyVolume %s
inputStartingSeedsLabelMapVolume: (an existing file name)
    Required: input starting seeds LabelMap image file name
    flag: --inputStartingSeedsLabelMapVolume %s
inputTensorVolume: (an existing file name)
    Required: input tensor image file name
    flag: --inputTensorVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputCostVolume: (a boolean or a file name)
    Output vcl_cost image
    flag: --outputCostVolume %s
outputSpeedVolume: (a boolean or a file name)
    Output speed image
    flag: --outputSpeedVolume %s
seedThreshold: (a float)
    Anisotropy threshold used for seed selection
    flag: --seedThreshold %f
startingSeedsLabel: (an integer (int or long))
    Label value for Starting Seeds
    flag: --startingSeedsLabel %d
stoppingValue: (a float)
```

```

Terminating value for vcl_cost function estimation
flag: --stoppingValue %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputCostVolume: (an existing file name)
    Output vcl_cost image
outputSpeedVolume: (an existing file name)
    Output speed image

```

gtractCreateGuideFiber[Link to code](#)Wraps command **** gtractCreateGuideFiber ****

title: Create Guide Fiber

category: Diffusion.GTRACT

description: This program will create a guide fiber by averaging fibers from a previously generated tract.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputFiber: (an existing file name)
    Required: input fiber tract file name
    flag: --inputFiber %s
numberOfPoints: (an integer (int or long))
    Number of points in output guide fiber
    flag: --numberOfPoints %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputFiber: (a boolean or a file name)
    Required: output guide fiber file name
    flag: --outputFiber %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display

```

```

    output, `file` - writes output to file, `none` - output is ignored
writeXMLPolyDataFile: (a boolean)
    Flag to make use of XML files when reading and writing vtkPolyData.
    flag: --writeXMLPolyDataFile

```

Outputs:

```

outputFiber: (an existing file name)
    Required: output guide fiber file name

```

gtractFastMarchingTracking[Link to code](#)

Wraps command **** gtractFastMarchingTracking ****

title: Fast Marching Tracking

category: Diffusion.GTRACT

description: This program will use a fast marching fiber tracking algorithm to identify fiber tracts from a tensor image. This program is the second portion of the algorithm. The user must first run gtractCostFastMarching to generate the vcl_cost image. The second step of the algorithm implemented here is a gradient descent solution from the defined ending region back to the seed points specified in gtractCostFastMarching. This algorithm is roughly based on the work by G. Parker et al. from IEEE Transactions On Medical Imaging, 21(5): 505-512, 2002. An additional feature of including anisotropy into the vcl_cost function calculation is included.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris. The original code here was developed by Daisy Espino.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
costStepSize: (a float)
    Cost image sub-voxel sampling
    flag: --costStepSize %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputAnisotropyVolume: (an existing file name)
    Required: input anisotropy image file name
    flag: --inputAnisotropyVolume %s
inputCostVolume: (an existing file name)
    Required: input vcl_cost image file name
    flag: --inputCostVolume %s
inputStartingSeedsLabelMapVolume: (an existing file name)
    Required: input starting seeds LabelMap image file name
    flag: --inputStartingSeedsLabelMapVolume %s

```

```

inputTensorVolume: (an existing file name)
    Required: input tensor image file name
    flag: --inputTensorVolume %s
maximumStepSize: (a float)
    Maximum step size to move when tracking
    flag: --maximumStepSize %f
minimumStepSize: (a float)
    Minimum step size to move when tracking
    flag: --minimumStepSize %f
numberOfIterations: (an integer (int or long))
    Number of iterations used for the optimization
    flag: --numberOfIterations %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputTract: (a boolean or a file name)
    Required: name of output vtkPolydata file containing tract lines and
    the point data collected along them.
    flag: --outputTract %s
seedThreshold: (a float)
    Anisotropy threshold used for seed selection
    flag: --seedThreshold %f
startingSeedsLabel: (an integer (int or long))
    Label value for Starting Seeds
    flag: --startingSeedsLabel %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
trackingThreshold: (a float)
    Anisotropy threshold used for fiber tracking
    flag: --trackingThreshold %f
writeXMLPolyDataFile: (a boolean)
    Flag to make use of the XML format for vtkPolyData fiber tracts.
    flag: --writeXMLPolyDataFile

```

Outputs:

```

outputTract: (an existing file name)
    Required: name of output vtkPolydata file containing tract lines and
    the point data collected along them.

```

gtractFiberTracking[Link to code](#)Wraps command **** gtractFiberTracking ****

title: Fiber Tracking

category: Diffusion.GTRACT

description: This program implements four fiber tracking methods (Free, Streamline, GraphSearch, Guided). The output of the fiber tracking is vtkPolyData (i.e. Polylines) that can be loaded into Slicer3 for visualization. The poly data can be saved in either old VTK format files (.vtk) or in the new VTK XML format (.xml). The polylines contain point data that defines ther Tensor at each point along the fiber tract. This can then be used to rendered as glyphs in Slicer3 and can be used to define several scalar measures without referencing back to the anisotropy images. (1) Free tracking is a basic streamlines algorithm. This is a direct implementation of the method original proposed by Bassar et al. The tracking follows the primarty eigenvector. The tracking begins with seed points in the starting region. Only those voxels above the specified anisotropy threshold in the starting region are used as seed points. Tracking terminates either as a result of maximum fiber length,

low anisotropy, or large curvature. This is a great way to explore your data. (2) The streamlines algorithm is a direct implementation of the method originally proposed by Basser et al. The tracking follows the primary eigenvector. The tracking begins with seed points in the starting region. Only those voxels above the specified anisotropy threshold in the starting region are used as seed points. Tracking terminates either by reaching the ending region or reaching some stopping criteria. Stopping criteria are specified using the following parameters: tracking threshold, curvature threshold, and max length. Only paths terminating in the ending region are kept in this method. The TEND algorithm proposed by Lazar et al. (Human Brain Mapping 18:306-321, 2003) has been instrumented. This can be enabled using the `-useTend` option while performing Streamlines tracking. This utilizes the entire diffusion tensor to deflect the incoming vector instead of simply following the primary eigenvector. The TEND parameters are set using the `-tendF` and `-tendG` options. (3) Graph Search tracking is the first step in the full GTRACT algorithm developed by Cheng et al. (NeuroImage 31(3): 1075-1085, 2006) for finding the tracks in a tensor image. This method was developed to generate fibers in a Tensor representation where crossing fibers occur. The graph search algorithm follows the primary eigenvector in non-ambiguous regions and utilizes branching and a graph search algorithm in ambiguous regions. Ambiguous tracking regions are defined based on two criteria: Branching AI Threshold (anisotropy values below this value and above the tracking threshold) and Curvature Major Eigen (angles of the primary eigenvector direction and the current tracking direction). In regions that meet this criteria, two or three tracking paths are considered. The first is the standard primary eigenvector direction. The second is the secondary eigenvector direction. This is based on the assumption that these regions may be prolate regions. If the Random Walk option is selected then a third direction is also considered. This direction is defined by a cone pointing from the current position to the centroid of the ending region. The interior angle of the cone is specified by the user with the Branch/Guide Angle parameter. A vector contained inside of the cone is selected at random and used as the third direction. This method can also utilize the TEND option where the primary tracking direction is that specified by the TEND method instead of the primary eigenvector. The parameter `'-maximumBranchPoints'` allows the tracking to have this number of branches being considered at a time. If this number of branch points is exceeded at any time, then the algorithm will revert back to a streamline algorithm until the number of branches is reduced. This allows the user to constrain the computational complexity of the algorithm. (4) The second phase of the GTRACT algorithm is Guided Tracking. This method incorporates anatomical information about the track orientation using an initial guess of the fiber track. In the originally proposed GTRACT method, this would be created from the fibers resulting from the Graph Search tracking. However, in practice this can be created using any method and could be defined manually. To create the guide fiber the program `gtractCreateGuideFiber` can be used. This program will load a fiber tract that has been generated and create a centerline representation of the fiber tract (i.e. a single fiber). In this method, the fiber tracking follows the primary eigenvector direction unless it deviates from the guide fiber track by an angle greater than that specified by the `'-guidedCurvatureThreshold'` parameter. The user must specify the guide fiber when running this program.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta, Greg Harris and Yongqiang Zhao.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
branchingAngle: (a float)
    Branching angle in degrees (recommended for GraphSearch fiber
    tracking method)
    flag: --branchingAngle %f
branchingThreshold: (a float)
    Anisotropy Branching threshold (recommended for GraphSearch fiber
```

```

        tracking method)
        flag: --branchingThreshold %f
curvatureThreshold: (a float)
    Curvature threshold in degrees (recommended for Free fiber tracking)
    flag: --curvatureThreshold %f
endingSeedsLabel: (an integer (int or long))
    Label value for Ending Seeds (required if Label number used to
    create seed point in Slicer was not 1)
    flag: --endingSeedsLabel %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
guidedCurvatureThreshold: (a float)
    Guided Curvature Threshold (Degrees)
    flag: --guidedCurvatureThreshold %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputAnisotropyVolume: (an existing file name)
    Required (for Free, Streamline, GraphSearch, and Guided fiber
    tracking methods): input anisotropy image file name
    flag: --inputAnisotropyVolume %s
inputEndingSeedsLabelMapVolume: (an existing file name)
    Required (for Streamline, GraphSearch, and Guided fiber tracking
    methods): input ending seeds LabelMap image file name
    flag: --inputEndingSeedsLabelMapVolume %s
inputStartingSeedsLabelMapVolume: (an existing file name)
    Required (for Free, Streamline, GraphSearch, and Guided fiber
    tracking methods): input starting seeds LabelMap image file name
    flag: --inputStartingSeedsLabelMapVolume %s
inputTensorVolume: (an existing file name)
    Required (for Free, Streamline, GraphSearch, and Guided fiber
    tracking methods): input tensor image file name
    flag: --inputTensorVolume %s
inputTract: (an existing file name)
    Required (for Guided fiber tracking method): guide fiber in
    vtkPolydata file containing one tract line.
    flag: --inputTract %s
maximumBranchPoints: (an integer (int or long))
    Maximum branch points (recommended for GraphSearch fiber tracking
    method)
    flag: --maximumBranchPoints %d
maximumGuideDistance: (a float)
    Maximum distance for using the guide fiber direction
    flag: --maximumGuideDistance %f
maximumLength: (a float)
    Maximum fiber length (voxels)
    flag: --maximumLength %f
minimumLength: (a float)
    Minimum fiber length. Helpful for filtering invalid tracts.
    flag: --minimumLength %f
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputTract: (a boolean or a file name)
    Required (for Free, Streamline, GraphSearch, and Guided fiber
    tracking methods): name of output vtkPolydata file containing tract

```

```

        lines and the point data collected along them.
        flag: --outputTract %s
randomSeed: (an integer (int or long))
        Random number generator seed
        flag: --randomSeed %d
seedThreshold: (a float)
        Anisotropy threshold for seed selection (recommended for Free fiber
        tracking)
        flag: --seedThreshold %f
startingSeedsLabel: (an integer (int or long))
        Label value for Starting Seeds (required if Label number used to
        create seed point in Slicer was not 1)
        flag: --startingSeedsLabel %d
stepSize: (a float)
        Fiber tracking step size
        flag: --stepSize %f
tendF: (a float)
        Tend F parameter
        flag: --tendF %f
tendG: (a float)
        Tend G parameter
        flag: --tendG %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
trackingMethod: ('Guided' or 'Free' or 'Streamline' or 'GraphSearch')
        Fiber tracking Filter Type: Guided|Free|Streamline|GraphSearch
        flag: --trackingMethod %s
trackingThreshold: (a float)
        Anisotropy threshold for fiber tracking (anisotropy values of the
        next point along the path)
        flag: --trackingThreshold %f
useLoopDetection: (a boolean)
        Flag to make use of loop detection.
        flag: --useLoopDetection
useRandomWalk: (a boolean)
        Flag to use random walk.
        flag: --useRandomWalk
useTend: (a boolean)
        Flag to make use of Tend F and Tend G parameters.
        flag: --useTend
writeXMLPolyDataFile: (a boolean)
        Flag to make use of the XML format for vtkPolyData fiber tracts.
        flag: --writeXMLPolyDataFile

```

Outputs:

```

outputTract: (an existing file name)
        Required (for Free, Streamline, GraphSearch, and Guided fiber
        tracking methods): name of output vtkPolydata file containing tract
        lines and the point data collected along them.

```

gtractImageConformity[Link to code](#)Wraps command **** gtractImageConformity ****

title: Image Conformity

category: Diffusion.GTRACT

description: This program will straighten out the Direction and Origin to match the Reference Image.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputReferenceVolume: (an existing file name)
    Required: input file containing the standard image to clone the
    characteristics of.
    flag: --inputReferenceVolume %s
inputVolume: (an existing file name)
    Required: input file containing the signed short image to reorient
    without resampling.
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output Nrrd or Nifti file containing the
    reoriented image in reference image space.
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output Nrrd or Nifti file containing the
    reoriented image in reference image space.
```

gtractInvertBSplineTransform

[Link to code](#)

Wraps command `** gtractInvertBSplineTransform **`

title: B-Spline Transform Inversion

category: Diffusion.GTRACT

description: This program will invert a B-Spline transform using a thin-plate spline approximation.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputReferenceVolume: (an existing file name)
    Required: input image file name to exemplify the anatomical space to
    interpolate over.
    flag: --inputReferenceVolume %s
inputTransform: (an existing file name)
    Required: input B-Spline transform file name
    flag: --inputTransform %s
landmarkDensity: (a list of items which are an integer (int or long))
    Number of landmark subdivisions in all 3 directions
    flag: --landmarkDensity %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputTransform: (a boolean or a file name)
    Required: output transform file name
    flag: --outputTransform %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputTransform: (an existing file name)
    Required: output transform file name
```

gtractInvertDisplacementField

[Link to code](#)

Wraps command **** gtractInvertDisplacementField ****

title: Invert Displacement Field

category: Diffusion.GTRACT

description: This program will invert a deformation field. The size of the deformation field is defined by an example image provided by the user

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
baseImage: (an existing file name)
    Required: base image used to define the size of the inverse field
    flag: --baseImage %s
deformationImage: (an existing file name)
    Required: Displacement field image
    flag: --deformationImage %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: Output deformation field
    flag: --outputVolume %s
subsamplingFactor: (an integer (int or long))
    Subsampling factor for the deformation field
    flag: --subsamplingFactor %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: Output deformation field
```

gtractInvertRigidTransform

[Link to code](#)

Wraps command `** gtractInvertRigidTransform **`

title: Rigid Transform Inversion

category: Diffusion.GTRACT

description: This program will invert a Rigid transform.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputTransform: (an existing file name)
    Required: input rigid transform file name
    flag: --inputTransform %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputTransform: (a boolean or a file name)
    Required: output transform file name
    flag: --outputTransform %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputTransform: (an existing file name)
    Required: output transform file name
```

gtractResampleAnisotropy[Link to code](#)

Wraps command **** gtractResampleAnisotropy ****

title: Resample Anisotropy

category: Diffusion.GTRACT

description: This program will resample a floating point image using either the Rigid or B-Spline transform. You may want to save the aligned B0 image after each of the anisotropy map co-registration steps with the anatomical image to check the registration quality with another tool.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
```

```

    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputAnatomicalVolume: (an existing file name)
    Required: input file containing the anatomical image whose
    characteristics will be cloned.
    flag: --inputAnatomicalVolume %s
inputAnisotropyVolume: (an existing file name)
    Required: input file containing the anisotropy image
    flag: --inputAnisotropyVolume %s
inputTransform: (an existing file name)
    Required: input Rigid OR Bspline transform file name
    flag: --inputTransform %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the resampled
    transformed anisotropy image.
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformType: ('Rigid' or 'B-Spline')
    Transform type: Rigid, B-Spline
    flag: --transformType %s

```

Outputs:

```

outputVolume: (an existing file name)
    Required: name of output NRRD file containing the resampled
    transformed anisotropy image.

```

gtractResampleB0[Link to code](#)Wraps command **** gtractResampleB0 ****

title: Resample B0

category: Diffusion.GTRACT

description: This program will resample a signed short image using either a Rigid or B-Spline transform. The user must specify a template image that will be used to define the origin, orientation, spacing, and size of the resampled image.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

[Mandatory]

[Optional]

args: (a unicode string)

```

    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputAnatomicalVolume: (an existing file name)
    Required: input file containing the anatomical image defining the
    origin, spacing and size of the resampled image (template)
    flag: --inputAnatomicalVolume %s
inputTransform: (an existing file name)
    Required: input Rigid OR Bspline transform file name
    flag: --inputTransform %s
inputVolume: (an existing file name)
    Required: input file containing the 4D image
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the resampled input
    image.
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformType: ('Rigid' or 'B-Spline')
    Transform type: Rigid, B-Spline
    flag: --transformType %s
vectorIndex: (an integer (int or long))
    Index in the diffusion weighted image set for the B0 image
    flag: --vectorIndex %d

```

Outputs:

```

outputVolume: (an existing file name)
    Required: name of output NRRD file containing the resampled input
    image.

```

gtractResampleCodeImage[Link to code](#)Wraps command **** gtractResampleCodeImage ****

title: Resample Code Image

category: Diffusion.GTRACT

description: This program will resample a short integer code image using either the Rigid or Inverse-B-Spline transform. The reference image is the DTI tensor anisotropy image space, and the input code image is in anatomical space.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS

R01NS050568-01A2S1

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputCodeVolume: (an existing file name)
    Required: input file containing the code image
    flag: --inputCodeVolume %s
inputReferenceVolume: (an existing file name)
    Required: input file containing the standard image to clone the
    characteristics of.
    flag: --inputReferenceVolume %s
inputTransform: (an existing file name)
    Required: input Rigid or Inverse-B-Spline transform file name
    flag: --inputTransform %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the resampled code
    image in acquisition space.
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformType: ('Rigid' or 'Affine' or 'B-Spline' or 'Inverse-B-
    Spline' or 'None')
    Transform type: Rigid or Inverse-B-Spline
    flag: --transformType %s

```

Outputs:

```

outputVolume: (an existing file name)
    Required: name of output NRRD file containing the resampled code
    image in acquisition space.

```

gtractResampleDWIInPlace

[Link to code](#)

Wraps command **** gtractResampleDWIInPlace ****

title: Resample DWI In Place

category: Diffusion.GTRACT

description: Resamples DWI image to structural image.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta, Greg Harris, Hans Johnson, and Joy Matsui.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debugLevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
    flag: --debugLevel %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
imageOutputSize: (a list of items which are an integer (int or long))
    The voxel lattice for the output image, padding is added if
    necessary. NOTE: if 0,0,0, then the inputVolume size is used.
    flag: --imageOutputSize %s
inputTransform: (an existing file name)
    Required: transform file derived from rigid registration of b0 image
    to reference structural image.
    flag: --inputTransform %s
inputVolume: (an existing file name)
    Required: input image is a 4D NRRD image.
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputResampledB0: (a boolean or a file name)
    Convenience function for extracting the first index location
    (assumed to be the B0)
    flag: --outputResampledB0 %s
outputVolume: (a boolean or a file name)
    Required: output image (NRRD file) that has been rigidly transformed
    into the space of the structural image and padded if image padding
    was changed from 0,0,0 default.
    flag: --outputVolume %s
referenceVolume: (an existing file name)
    If provided, resample to the final space of the referenceVolume 3D
    data set.
    flag: --referenceVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
warpDWITransform: (an existing file name)
    Optional: transform file to warp gradient volumes.
    flag: --warpDWITransform %s
```

Outputs:


```

outputResampledB0: (an existing file name)
    Convenience function for extracting the first index location
    (assumed to be the B0)
outputVolume: (an existing file name)
    Required: output image (NRRD file) that has been rigidly transformed
    into the space of the structural image and padded if image padding
    was changed from 0,0,0 default.

```

gtractResampleFibers

[Link to code](#)

Wraps command **** gtractResampleFibers ****

title: Resample Fibers

category: Diffusion.GTRACT

description: This program will resample a fiber tract with respect to a pair of deformation fields that represent the forward and reverse deformation fields.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputForwardDeformationFieldVolume: (an existing file name)
    Required: input forward deformation field image file name
    flag: --inputForwardDeformationFieldVolume %s
inputReverseDeformationFieldVolume: (an existing file name)
    Required: input reverse deformation field image file name
    flag: --inputReverseDeformationFieldVolume %s
inputTract: (an existing file name)
    Required: name of input vtkPolydata file containing tract lines.
    flag: --inputTract %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputTract: (a boolean or a file name)
    Required: name of output vtkPolydata file containing tract lines and
    the point data collected along them.
    flag: --outputTract %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display

```

```

    output, `file` - writes output to file, `none` - output is ignored
writeXMLPolyDataFile: (a boolean)
    Flag to make use of the XML format for vtkPolyData fiber tracts.
    flag: --writeXMLPolyDataFile

```

Outputs:

```

outputTract: (an existing file name)
    Required: name of output vtkPolydata file containing tract lines and
    the point data collected along them.

```

gtractTensor[Link to code](#)Wraps command **** gtractTensor ****

title: Tensor Estimation

category: Diffusion.GTRACT

description: This step will convert a b-value averaged diffusion tensor image to a 3x3 tensor voxel image. This step takes the diffusion tensor image data and generates a tensor representation of the data based on the signal intensity decay, b values applied, and the diffusion difrections. The apparent diffusion coefficient for a given orientation is computed on a pixel-by-pixel basis by fitting the image data (voxel intensities) to the Stejskal-Tanner equation. If at least 6 diffusion directions are used, then the diffusion tensor can be computed. This program uses itk::DiffusionTensor3DReconstructionImageFilter. The user can adjust background threshold, median filter, and isotropic resampling.

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta and Greg Harris.

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```

[Mandatory]

[Optional]
applyMeasurementFrame: (a boolean)
    Flag to apply the measurement frame to the gradient directions
    flag: --applyMeasurementFrame
args: (a unicode string)
    Additional parameters to the command
    flag: %s
b0Index: (an integer (int or long))
    Index in input vector index to extract
    flag: --b0Index %d
backgroundSuppressingThreshold: (an integer (int or long))
    Image threshold to suppress background. This sets a threshold used
    on the b0 image to remove background voxels from processing.
    Typically, values of 100 and 500 work well for Siemens and GE DTI
    data, respectively. Check your data particularly in the globus
    pallidus to make sure the brain tissue is not being eliminated with
    this threshold.
    flag: --backgroundSuppressingThreshold %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```

```

ignoreIndex: (a list of items which are an integer (int or long))
    Ignore diffusion gradient index. Used to remove specific gradient
    directions with artifacts.
    flag: --ignoreIndex %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Required: input image 4D NRRD image. Must contain data based on at
    least 6 distinct diffusion directions. The inputVolume is allowed to
    have multiple b0 and gradient direction images. Averaging of the b0
    image is done internally in this step. Prior averaging of the DWIs
    is not required.
    flag: --inputVolume %s
maskProcessingMode: ('NOMASK' or 'ROIAUTO' or 'ROI')
    ROIAUTO: mask is implicitly defined using a otsu foreground and hole
    filling algorithm. ROI: Uses the masks to define what parts of the
    image should be used for computing the transform. NOMASK: no mask
    used
    flag: --maskProcessingMode %s
maskVolume: (an existing file name)
    Mask Image, if maskProcessingMode is ROI
    flag: --maskVolume %s
medianFilterSize: (a list of items which are an integer (int or
    long))
    Median filter radius in all 3 directions
    flag: --medianFilterSize %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the Tensor vector
    image
    flag: --outputVolume %s
resampleIsotropic: (a boolean)
    Flag to resample to isotropic voxels. Enabling this feature is
    recommended if fiber tracking will be performed.
    flag: --resampleIsotropic
size: (a float)
    Isotropic voxel size to resample to
    flag: --size %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Required: name of output NRRD file containing the Tensor vector
    image

```

gtractTransformToDisplacementField[Link to code](#)Wraps command **** gtractTransformToDisplacementField ****

title: Create Displacement Field

category: Diffusion.GTRACT

description: This program will compute forward deformation from the given Transform. The size of the DF is equal to MNI space

version: 4.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:GTRACT>

license: <http://mri.radiology.uiowa.edu/copyright/GTRACT-Copyright.txt>

contributor: This tool was developed by Vincent Magnotta, Madhura Ing  alhalikar, and Greg Harris

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputReferenceVolume: (an existing file name)
    Required: input image file name to exemplify the anatomical space
    over which to vcl_express the transform as a displacement field.
    flag: --inputReferenceVolume %s
inputTransform: (an existing file name)
    Input Transform File Name
    flag: --inputTransform %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputDeformationFieldVolume: (a boolean or a file name)
    Output deformation field
    flag: --outputDeformationFieldVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputDeformationFieldVolume: (an existing file name)
    Output deformation field
```

interfaces.semtools.diffusion.maxcurvature

maxcurvature

[Link to code](#)

Wraps command **** maxcurvature ****

title: MaxCurvature-Hessian (DTIProcess)

category: Diffusion

description: This program computes the Hessian of the FA image (  image). We use this scalar image as a registration input when doing DTI atlas building. For most adult FA we use a sigma of 2 whereas for neonate or primate images and sigma of 1 or 1.5 is more appropriate. For really noisy images, 2.5 - 4 can be considered.

The final image (–output) shows the main feature of the input image.

version: 1.1.0

documenta­tion-url: <http://www.slicer.org/slicerWiki/index.php/Documenta­tion/Nightly/Extensions/DTIProcess>

license: Copyright (c) Casey Goodlett. All rights reserved.

See <http://www.ia.unc.edu/dev/Copyright.htm> for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett

acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); (1=University of Iowa Department of Psychiatry, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering) provided conversions to make DTIProcess compatible with Slicer execution, and simplified the stand-alone build requirements by removing the dependancies on boost and a fortran compiler.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
image: (an existing file name)
    FA Image
    flag: --image %s
output: (a boolean or a file name)
    Output File
    flag: --output %s
sigma: (a float)
    Scale of Gradients
    flag: --sigma %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    produce verbose output
    flag: --verbose
```

Outputs:

```
output: (an existing file name)
    Output File
```

interfaces.semtools.diffusion.tractography.commandlineonly

fiberstats

[Link to code](#)

Wraps command ** fiberstats **

title: FiberStats (DTIProcess)

category: Diffusion.Tractography.CommandLineOnly

description: Obsolete tool - Not used anymore

version: 1.1.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Extensions/DTIPProcess>

license: Copyright (c) Casey Goodlett. All rights reserved.

See <http://www.ia.unc.edu/dev/Copyright.htm> for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett

acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); (1=University of Iowa Department of Psychiatry, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering) provided conversions to make DTIPProcess compatible with Slicer execution, and simplified the stand-alone build requirements by removing the dependencies on boost and a fortran compiler.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyype default value: {})
    Environment variables
fiber_file: (an existing file name)
    DTI Fiber File
    flag: --fiber_file %s
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    produce verbose output
    flag: --verbose
```

Outputs:

```
None
```

interfaces.semtools.diffusion.tractography.fiberprocess

fiberprocess

[Link to code](#)

Wraps command **** fiberprocess ****

title: FiberProcess (DTIPProcess)

category: Diffusion.Tractography

description: fiberprocess is a tool that manage fiber files extracted from the fibertrack tool or any fiber tracking algorithm. It takes as an input .fib and .vtk files (`-fiber_file`) and saves the changed fibers (`-fiber_output`) into the 2 same formats. The main purpose of this tool is to deform the fiber file with a transformation field as an input (`-displacement_field` or `-h_field` depending if you deal with dfield or hfield). To use that option you need to specify the tensor field from which the fiber file was extracted with the option `-tensor_volume`. The transformation applied on the fiber file is the inverse of the one input. If the transformation is from one case to

an atlas, fiberprocess assumes that the fiber file is in the atlas space and you want it in the original case space, so it's the inverse of the transformation which has been computed. You have 2 options for fiber modification. You can either deform the fibers (their geometry) into the space OR you can keep the same geometry but map the diffusion properties (fa, md, lbd's...) of the original tensor field along the fibers at the corresponding locations. This is triggered by the `--no_warp` option. To use the previous example: when you have a tensor field in the original space and the deformed tensor field in the atlas space, you want to track the fibers in the atlas space, keeping this geometry but with the original case diffusion properties. Then you can specify the transformations field (from original case -> atlas) and the original tensor field with the `--tensor_volume` option. With fiberprocess you can also binarize a fiber file. Using the `--voxelize` option will create an image where each voxel through which a fiber is passing is set to 1. The output is going to be a binary image with the values 0 or 1 by default but the 1 value voxel can be set to any number with the `--voxel_label` option. Finally you can create an image where the value at the voxel is the number of fiber passing through. (`--voxelize_count_fibers`)

version: 1.0.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Extensions/DTIPProcess>

license: Copyright (c) Casey Goodlett. All rights reserved. See <http://www.ia.unc.edu/dev/Copyright.htm> for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
displacement_field: (an existing file name)
    Displacement Field for warp and statistics lookup. If this option is
    used tensor-volume must also be specified.
    flag: --displacement_field %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
fiber_file: (an existing file name)
    DTI fiber file
    flag: --fiber_file %s
fiber_output: (a boolean or a file name)
    Output fiber file. May be warped or updated with new data depending
    on other options used.
    flag: --fiber_output %s
fiber_radius: (a float)
    set radius of all fibers to this value
    flag: --fiber_radius %f
h_field: (an existing file name)
    HField for warp and statistics lookup. If this option is used
    tensor-volume must also be specified.
    flag: --h_field %s
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
index_space: (a boolean)
    Use index-space for fiber output coordinates, otherwise us world
    space for fiber output coordinates (from tensor file).
    flag: --index_space
noDataChange: (a boolean)
```

```

    Do not change data ???
    flag: --noDataChange
no_warp: (a boolean)
    Do not warp the geometry of the tensors only obtain the new
    statistics.
    flag: --no_warp
saveProperties: (a boolean)
    save the tensor property as scalar data into the vtk (only works for
    vtk fiber files).
    flag: --saveProperties
tensor_volume: (an existing file name)
    Interpolate tensor values from the given field
    flag: --tensor_volume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    produce verbose output
    flag: --verbose
voxel_label: (an integer (int or long))
    Label for voxelized fiber
    flag: --voxel_label %d
voxelize: (a boolean or a file name)
    Voxelize fiber into a label map (the labelmap filename is the
    argument of -V). The tensor file must be specified using -T for
    information about the size, origin, spacing of the image. The
    deformation is applied before the voxelization
    flag: --voxelize %s
voxelize_count_fibers: (a boolean)
    Count number of fibers per-voxel instead of just setting to 1
    flag: --voxelize_count_fibers

```

Outputs:

```

fiber_output: (an existing file name)
    Output fiber file. May be warped or updated with new data depending
    on other options used.
voxelize: (an existing file name)
    Voxelize fiber into a label map (the labelmap filename is the
    argument of -V). The tensor file must be specified using -T for
    information about the size, origin, spacing of the image. The
    deformation is applied before the voxelization

```

interfaces.semtools.diffusion.tractography.fibertrack**fibertrack**[Link to code](#)Wraps command **** fibertrack ****

title: FiberTrack (DTIProcess)

category: Diffusion.Tractography

description: This program implements a simple streamline tractography method based on the principal eigenvector of the tensor field. A fourth order Runge-Kutta integration rule used to advance the streamlines. As a first parameter you have to input the tensor field (with the `-input_tensor_file` option). Then the region of interest image file is set with the `-input_roi_file`. Next you want to set the output fiber file name after the `-output_fiber_file` option. You can specify the label value in the `input_roi_file` with the `-target_label`, -

source_label and –forbidden_label options. By default target label is 1, source label is 2 and forbidden label is 0. The source label is where the streamlines are seeded, the target label defines the voxels through which the fibers must pass by to be kept in the final fiber file and the forbidden label defines the voxels where the streamlines are stopped if they pass through it. There is also a –whole_brain option which, if enabled, consider both target and source labels of the roi image as target labels and all the voxels of the image are considered as sources. During the tractography, the –fa_min parameter is used as the minimum value needed at different voxel for the tracking to keep going along a streamline. The –step_size parameter is used for each iteration of the tracking algorithm and defines the length of each step. The –max_angle option defines the maximum angle allowed between two successive segments along the tracked fiber.

version: 1.1.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Extensions/DTIPProcess>

license: Copyright (c) Casey Goodlett. All rights reserved.

See <http://www.ia.unc.edu/dev/Copyright.htm> for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett

acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); (1=University of Iowa Department of Psychiatry, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering) provided conversions to make DTIPProcess compatible with Slicer execution, and simplified the stand-alone build requirements by removing the dependencies on boost and a fortran compiler.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
forbidden_label: (an integer (int or long))
    Forbidden label
    flag: --forbidden_label %d
force: (a boolean)
    Ignore sanity checks.
    flag: --force
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
input_roi_file: (an existing file name)
    The filename of the image which contains the labels used for seeding
    and constraining the algorithm.
    flag: --input_roi_file %s
input_tensor_file: (an existing file name)
    Tensor Image
    flag: --input_tensor_file %s
max_angle: (a float)
    Maximum angle of change in radians
    flag: --max_angle %f
min_fa: (a float)
    The minimum FA threshold to continue tractography
    flag: --min_fa %f
output_fiber_file: (a boolean or a file name)
    The filename for the fiber file produced by the algorithm. This file
    must end in a .fib or .vtk extension for ITK spatial object and
```

```

        vtkPolyData formats respectively.
        flag: --output_fiber_file %s
really_verbose: (a boolean)
        Follow detail of fiber tracking algorithm
        flag: --really_verbose
source_label: (an integer (int or long))
        The label of voxels in the labelfile to use for seeding
        tractography. One tract is seeded from the center of each voxel with
        this label
        flag: --source_label %d
step_size: (a float)
        Step size in mm for the tracking algorithm
        flag: --step_size %f
target_label: (an integer (int or long))
        The label of voxels in the labelfile used to constrain tractography.
        Tracts that do not pass through a voxel with this label are
        rejected. Set this keep all tracts.
        flag: --target_label %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
        produce verbose output
        flag: --verbose
whole_brain: (a boolean)
        If this option is enabled all voxels in the image are used to seed
        tractography. When this option is enabled both source and target
        labels function as target labels
        flag: --whole_brain

```

Outputs:

```

output_fiber_file: (an existing file name)
        The filename for the fiber file produced by the algorithm. This file
        must end in a .fib or .vtk extension for ITK spatial object and
        vtkPolyData formats respectively.

```

interfaces.semtools.diffusion.tractography.ukftractography**UKFTractography**[Link to code](#)Wraps command **** UKFTractography ****

title: UKF Tractography

category: Diffusion.Tractography

description: This module traces fibers in a DWI Volume using the multiple tensor unscented Kalman Filter methodology. For more informations check the documentation.

version: 1.0

documentation-url: <http://www.nitrc.org/plugins/mwiki/index.php/ukftractography:MainPage>

contributor: Yogesh Rath, Stefan Lienhard, Yinpeng Li, Martin Styner, Ipek Oguz, Yundi Shi, Christian Baumgartner, Kent Williams, Hans Johnson, Peter Savadjiev, Carl-Fredrik Westin.

acknowledgements: The development of this module was supported by NIH grants R01 MH097979 (PI Rath), R01 MH092862 (PIs Westin and Verma), U01 NS083223 (PI Westin), R01 MH074794 (PI Westin) and P41 EB015902 (PI Kikinis).

Inputs:

```

[Mandatory]

[Optional]
Q1: (a float)
    Process noise for eigenvalues
    flag: --Q1 %f
Qm: (a float)
    Process noise for angles/direction
    flag: --Qm %f
Qw: (a float)
    Process noise for free water weights, ignored if no free water
    estimation
    flag: --Qw %f
Rs: (a float)
    Measurement noise
    flag: --Rs %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dwiFile: (an existing file name)
    Input DWI volume
    flag: --dwiFile %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
freeWater: (a boolean)
    Adds a term for free water difusion to the model. (Note for experts:
    if checked, the 1T simple model is forced)
    flag: --freeWater
fullTensorModel: (a boolean)
    Whether to use the full tensor model. If unchecked, use the default
    simple tensor model
    flag: --fullTensorModel
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
labels: (a list of items which are an integer (int or long))
    A vector of the ROI labels to be used
    flag: --labels %s
maskFile: (an existing file name)
    Mask for diffusion tractography
    flag: --maskFile %s
maxBranchingAngle: (a float)
    Maximum branching angle, in degrees. When using multiple tensors, a
    new branch will be created when the tensors' major directions form
    an angle between (minBranchingAngle, maxBranchingAngle). Branching
    is supressed when this maxBranchingAngle is set to 0.0
    flag: --maxBranchingAngle %f
maxHalfFiberLength: (a float)
    The max length limit of the half fibers generated during
    tractography. Here the fiber is 'half' because the tractography goes
    in only one direction from one seed point at a time
    flag: --maxHalfFiberLength %f
minBranchingAngle: (a float)
    Minimum branching angle, in degrees. When using multiple tensors, a
    new branch will be created when the tensors' major directions form
    an angle between (minBranchingAngle, maxBranchingAngle)

```

```
    flag: --minBranchingAngle %f
minFA: (a float)
    Abort the tractography when the Fractional Anisotropy is less than
    this value
    flag: --minFA %f
minGA: (a float)
    Abort the tractography when the Generalized Anisotropy is less than
    this value
    flag: --minGA %f
numTensor: ('1' or '2')
    Number of tensors used
    flag: --numTensor %s
numThreads: (an integer (int or long))
    Number of threads used during computation. Set to the number of
    cores on your workstation for optimal speed. If left undefined the
    number of cores detected will be used.
    flag: --numThreads %d
recordCovariance: (a boolean)
    Whether to store the covariance. Will generate field 'covariance' in
    fiber.
    flag: --recordCovariance
recordFA: (a boolean)
    Whether to store FA. Attaches field 'FA', and 'FA2' for 2-tensor
    case to fiber.
    flag: --recordFA
recordFreeWater: (a boolean)
    Whether to store the fraction of free water. Attaches field
    'FreeWater' to fiber.
    flag: --recordFreeWater
recordLength: (a float)
    Record length of tractography, in millimeters
    flag: --recordLength %f
recordNMSE: (a boolean)
    Whether to store NMSE. Attaches field 'NMSE' to fiber.
    flag: --recordNMSE
recordState: (a boolean)
    Whether to attach the states to the fiber. Will generate field
    'state'.
    flag: --recordState
recordTensors: (a boolean)
    Recording the tensors enables Slicer to color the fiber bundles by
    FA, orientation, and so on. The fields will be called 'TensorN',
    where N is the tensor number.
    flag: --recordTensors
recordTrace: (a boolean)
    Whether to store Trace. Attaches field 'Trace', and 'Trace2' for
    2-tensor case to fiber.
    flag: --recordTrace
seedFALimit: (a float)
    Seed points whose FA are below this value are excluded
    flag: --seedFALimit %f
seedsFile: (an existing file name)
    Seeds for diffusion. If not specified, full brain tractography will
    be performed, and the algorithm will start from every voxel in the
    brain mask where the Generalized Anisotropy is bigger than 0.18
    flag: --seedsFile %s
seedsPerVoxel: (an integer (int or long))
    Each seed generates a fiber, thus using more seeds generates more
```

```

        fibers. In general use 1 or 2 seeds, and for a more thorough result
        use 5 or 10 (depending on your machine this may take up to 2 days to
        run).,
        flag: --seedsPerVoxel %d
stepLength: (a float)
        Step length of tractography, in millimeters
        flag: --stepLength %f
storeGlyphs: (a boolean)
        Store tensors' main directions as two-point lines in a separate file
        named glyphs_{tracts}. When using multiple tensors, only the major
        tensors' main directions are stored
        flag: --storeGlyphs
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
tracts: (a boolean or a file name)
        Tracts generated, with first tensor output
        flag: --tracts %s
tractsWithSecondTensor: (a boolean or a file name)
        Tracts generated, with second tensor output (if there is one)
        flag: --tractsWithSecondTensor %s
writeAsciiTracts: (a boolean)
        Write tract file as a VTK binary data file
        flag: --writeAsciiTracts
writeUncompressedTracts: (a boolean)
        Write tract file as a VTK uncompressed data file
        flag: --writeUncompressedTracts

```

Outputs:

```

tracts: (an existing file name)
        Tracts generated, with first tensor output
tractsWithSecondTensor: (an existing file name)
        Tracts generated, with second tensor output (if there is one)

```

interfaces.semtools.featurecreator**GenerateCsfClippedFromClassifiedImage**

[Link to code](#)

Wraps command **** GenerateCsfClippedFromClassifiedImage ****

title: GenerateCsfClippedFromClassifiedImage

category: FeatureCreator

description: Get the distance from a voxel to the nearest voxel of a given tissue type.

version: 0.1.0.\$Revision: 1 \$(alpha)

documentation-url: <http://www.na-mic.org/>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was written by Hans J. Johnson.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
        Additional parameters to the command

```

```

    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputCassifiedVolume: (an existing file name)
    Required: input tissue label image
    flag: --inputCassifiedVolume %s
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Required: output image

```

interfaces.semtools.filtering.denoising

UnbiasedNonLocalMeans

[Link to code](#)

Wraps command `** UnbiasedNonLocalMeans **`

title: Unbiased NLM for MRI

category: Filtering.Denoising

description: This module implements a fast version of the popular Non-Local Means filter for image denoising. This algorithm

In the original formulation a patch with a certain radius is centered in each of the voxels, and the Mean Squared Error between each pair of corresponding voxels is computed. In this implementation, only the mean value and gradient components are compared. This, together with an efficient memory management, can attain a speed-up of nearly 20x. Besides, the filtering is more accurate than the original with poor SNR. This code is intended for its use with MRI (or any other Rician-distributed modality): the second order moment is estimated, then we subtract twice the squared power of noise, and finally we take the square root of the result to remove the Rician bias. The original implementation of the NLM filter may be found in: A. Buades, B. Coll, J. Morel, “A review of image denoising algorithms, with a new one”, *Multiscale Modelling and Simulation* 4(2): 490-530. 2005. The correction of the Rician bias is described in the following reference (among others): S. Aja-Fernandez, K. Krissian, “An unbiased Non-Local Means scheme for DWI filtering”, in: *Proceedings of the MICCAI Workshop on Computational Diffusion MRI*, 2008, pp. 277-284. The whole description of this version may be found in the following paper (please, cite it if you are willing to use this software): A. Tristan-Vega, V. Garcia Perez, S. Aja-Fernandez, and C.-F. Westin, “Efficient and Robust Nonlocal Means Denoising of MR Data Based on Salient Features Matching”, *Computer Methods and Programs in Biomedicine*. (Accepted for publication) 2011.

version: 0.0.1.\$Revision: 1 \$(beta)

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Modules:UnbiasedNonLocalMeans-Documentation-3.6>

contributor: Antonio Tristan Vega, Veronica Garcia-Perez, Santiago Aja-Fernandez, Carl-Fredrik Westin

acknowledgements: Supported by grant number FMECD-2010/71131616E from the Spanish Ministry of Education/Fulbright Committee

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
hp: (a float)
    This parameter is related to noise; the larger the parameter, the
    more aggressive the filtering. Should be near 1, and only values
    between 0.8 and 1.2 are allowed
    flag: --hp %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input MRI volume.
    flag: %s, position: -2
outputVolume: (a boolean or a file name)
    Output (filtered) MRI volume.
    flag: %s, position: -1
ps: (a float)
    To accelerate computations, preselection is used: if the normalized
    difference is above this threshold, the voxel will be discarded (non
    used for average)
    flag: --ps %f
rc: (a list of items which are an integer (int or long))
    Similarity between blocks is computed as the difference between mean
    values and gradients. These parameters are computed fitting a
    hyperplane with LS inside a neighborhood of this size
    flag: --rc %s
rs: (a list of items which are an integer (int or long))
    The algorithm search for similar voxels in a neighborhood of this
    radius (radii larger than 5,5,5 are very slow, and the results can
    be only marginally better. Small radii may fail to effectively
    remove the noise).
    flag: --rs %s
sigma: (a float)
    The root power of noise (sigma) in the complex Gaussian process the
    Rician comes from. If it is underestimated, the algorithm fails to
    remove the noise. If it is overestimated, over-blurring is likely to
    occur.
    flag: --sigma %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Output (filtered) MRI volume.

```

interfaces.semtools.filtering.featuredetection

CannyEdge

[Link to code](#)

Wraps command **** CannyEdge ****

title: Canny Edge Detection

category: Filtering.FeatureDetection

description: Get the distance from a voxel to the nearest voxel of a given tissue type.

version: 0.1.0.(alpha)

documentation-url: <http://www.na-mic.org/>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was written by Hans J. Johnson.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Required: input tissue label image
    flag: --inputVolume %s
lowerThreshold: (a float)
    Threshold is the lowest allowed value in the output image. Its data
    type is the same as the data type of the output image. Any values
    below the Threshold level will be replaced with the OutsideValue
    parameter value, whose default is zero.
    flag: --lowerThreshold %f
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
upperThreshold: (a float)
    Threshold is the lowest allowed value in the output image. Its data
    type is the same as the data type of the output image. Any values
    below the Threshold level will be replaced with the OutsideValue
    parameter value, whose default is zero.
    flag: --upperThreshold %f
variance: (a float)
    Variance and Maximum error are used in the Gaussian smoothing of the
    input image. See itkDiscreteGaussianImageFilter for information on
    these parameters.
    flag: --variance %f
```

Outputs:


```
outputVolume: (an existing file name)
    Required: output image
```

CannySegmentationLevelSetImageFilter

[Link to code](#)

Wraps command `** CannySegmentationLevelSetImageFilter **`

title: Canny Level Set Image Filter

category: Filtering.FeatureDetection

description: The CannySegmentationLevelSet is commonly used to refine a manually generated manual mask.

version: 0.3.0

license: CC

contributor: Regina Kim

acknowledgements: This command module was derived from Insight/Examples/Segmentation/CannySegmentationLevelSetImage (copyright) Insight Software Consortium. See http://wiki.na-mic.org/Wiki/index.php/Slicer3:Execution_Model_Documentation for more detailed descriptions.

Inputs:

```
[Mandatory]

[Optional]
advectionWeight: (a float)
    Controls the smoothness of the resulting mask, small number are more
    smooth, large numbers allow more sharp corners.
    flag: --advectionWeight %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cannyThreshold: (a float)
    Canny Threshold Value
    flag: --cannyThreshold %f
cannyVariance: (a float)
    Canny variance
    flag: --cannyVariance %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initialModel: (an existing file name)
    flag: --initialModel %s
initialModelIsovalue: (a float)
    The identification of the input model iso-surface. (for a binary
    image with 0s and 1s use 0.5) (for a binary image with 0s and 255's
    use 127.5).
    flag: --initialModelIsovalue %f
inputVolume: (an existing file name)
    flag: --inputVolume %s
maxIterations: (an integer (int or long))
    The
    flag: --maxIterations %d
outputSpeedVolume: (a boolean or a file name)
    flag: --outputSpeedVolume %s
outputVolume: (a boolean or a file name)
```

```
flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputSpeedVolume: (an existing file name)
outputVolume: (an existing file name)
```

DilateImage

[Link to code](#)

Wraps command **** DilateImage ****

title: Dilate Image

category: Filtering.FeatureDetection

description: Uses mathematical morphology to dilate the input images.

version: 0.1.0.\$Revision: 1 \$(alpha)

documentation-url: <http://www.na-mic.org/>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    flag: --inputMaskVolume %s
inputRadius: (an integer (int or long))
    Required: input neighborhood radius
    flag: --inputRadius %d
inputVolume: (an existing file name)
    Required: input image
    flag: --inputVolume %s
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

DilateMask

[Link to code](#)

Wraps command **** DilateMask ****

title: Dilate Image

category: Filtering.FeatureDetection

description: Uses mathematical morphology to dilate the input images.

version: 0.1.0.\$Revision: 1 \$(alpha)

documentation-url: <http://www.na-mic.org/>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputBinaryVolume: (an existing file name)
    Required: input brain mask image
    flag: --inputBinaryVolume %s
inputVolume: (an existing file name)
    Required: input image
    flag: --inputVolume %s
lowerThreshold: (a float)
    Required: lowerThreshold value
    flag: --lowerThreshold %f
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
sizeStructuralElement: (an integer (int or long))
    size of structural element. sizeStructuralElement=1 means that 3x3x3
    structuring element for 3D
    flag: --sizeStructuralElement %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

DistanceMaps

[Link to code](#)

Wraps command **** DistanceMaps ****

title: Mauerer Distance
category: Filtering.FeatureDetection
description: Get the distance from a voxel to the nearest voxel of a given tissue type.
version: 0.1.0.\$Revision: 1 \$(alpha)
documentation-url: <http://www.na-mic.org/>
license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>
contributor: This tool was developed by Mark Scully and Jeremy Bockholt.
Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputLabelVolume: (an existing file name)
    Required: input tissue label image
    flag: --inputLabelVolume %s
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    flag: --inputMaskVolume %s
inputTissueLabel: (an integer (int or long))
    Required: input integer value of tissue type used to calculate
    distance
    flag: --inputTissueLabel %d
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

DumpBinaryTrainingVectors

[Link to code](#)

Wraps command `** DumpBinaryTrainingVectors **`

title: Erode Image
category: Filtering.FeatureDetection
description: Uses mathematical morphology to erode the input images.
version: 0.1.0.\$Revision: 1 \$(alpha)
documentation-url: <http://www.na-mic.org/>
license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>
contributor: This tool was developed by Mark Scully and Jeremy Bockholt.
Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputHeaderFilename: (an existing file name)
    Required: input header file name
    flag: --inputHeaderFilename %s
inputVectorFilename: (an existing file name)
    Required: input vector filename
    flag: --inputVectorFilename %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

None

ErodeImage

[Link to code](#)

Wraps command **** ErodeImage ****

title: Erode Image

category: Filtering.FeatureDetection

description: Uses mathematical morphology to erode the input images.

version: 0.1.0.\$Revision: 1 \$(alpha)

documentation-url: <http://www.na-mic.org/>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    flag: --inputMaskVolume %s

```

```
inputRadius: (an integer (int or long))
    Required: input neighborhood radius
    flag: --inputRadius %d
inputVolume: (an existing file name)
    Required: input image
    flag: --inputVolume %s
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

FlippedDifference

[Link to code](#)

Wraps command **** FlippedDifference ****

title: Flip Image

category: Filtering.FeatureDetection

description: Difference between an image and the axially flipped version of that image.

version: 0.1.0.\$Revision: 1 \$(alpha)

documentation-url: <http://www.na-mic.org/>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    flag: --inputMaskVolume %s
inputVolume: (an existing file name)
    Required: input image
    flag: --inputVolume %s
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
```

```
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
Required: output image
```

GenerateBrainClippedImage[Link to code](#)Wraps command **** GenerateBrainClippedImage ****title: **GenerateBrainClippedImage**

category: Filtering.FeatureDetection

description: Automatic FeatureImages using neural networks

version: 1.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Eun Young Kim

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputImg: (an existing file name)
    input volume 1, usually t1 image
    flag: --inputImg %s
inputMsk: (an existing file name)
    input volume 2, usually t2 image
    flag: --inputMsk %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputFileName: (a boolean or a file name)
    (required) output file name
    flag: --outputFileName %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputFileName: (an existing file name)
    (required) output file name
```

GenerateSummedGradientImage[Link to code](#)

Wraps command **** GenerateSummedGradientImage ****

title: GenerateSummedGradient

category: Filtering.FeatureDetection

description: Automatic FeatureImages using neural networks

version: 1.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Greg Harris, Eun Young Kim

Inputs:

```
[Mandatory]

[Optional]
MaximumGradient: (a boolean)
    If set this flag, it will compute maximum gradient between two input
    volumes instead of sum of it.
    flag: --MaximumGradient
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume1: (an existing file name)
    input volume 1, usually t1 image
    flag: --inputVolume1 %s
inputVolume2: (an existing file name)
    input volume 2, usually t2 image
    flag: --inputVolume2 %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputFileName: (a boolean or a file name)
    (required) output file name
    flag: --outputFileName %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputFileName: (an existing file name)
    (required) output file name
```

GenerateTestImage

[Link to code](#)

Wraps command **** GenerateTestImage ****

title: DownSampleImage

category: Filtering.FeatureDetection

description: Down sample image for testing

version: 1.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Eun Young Kim

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyype default value: {})
    Environment variables
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    input volume 1, usally t1 image
    flag: --inputVolume %s
lowerBoundOfOutputVolume: (a float)
    flag: --lowerBoundOfOutputVolume %f
outputVolume: (a boolean or a file name)
    (required) output file name
    flag: --outputVolume %s
outputVolumeSize: (a float)
    output Volume Size
    flag: --outputVolumeSize %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
upperBoundOfOutputVolume: (a float)
    flag: --upperBoundOfOutputVolume %f
```

Outputs:

```
outputVolume: (an existing file name)
    (required) output file name
```

GradientAnisotropicDiffusionImageFilter[Link to code](#)**Wraps command** `** GradientAnisotropicDiffusionImageFilter **`**title:** GradientAnisopropicDiffusionFilter**category:** Filtering.FeatureDetection**description:** Image Smoothing using Gradient Anisotropic Diffuesion Filer**contributor:** This tool was developed by Eun Young Kim by modifying ITK Example**Inputs:**

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
conductance: (a float)
    Conductance for diffusion process
    flag: --conductance %f
environ: (a dictionary with keys which are a bytes or None or a value
```

```
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {}))
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Required: input image
    flag: --inputVolume %s
numberOfIterations: (an integer (int or long))
    Optional value for number of Iterations
    flag: --numberOfIterations %d
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timeStep: (a float)
    Time step for diffusion process
    flag: --timeStep %f
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

HammerAttributeCreator

[Link to code](#)

Wraps command **** HammerAttributeCreator ****

title: HAMMER Feature Vectors

category: Filtering.FeatureDetection

description: Create the feature vectors used by HAMMER.

version: 0.1.0.\$Revision: 1 \$(alpha)

documentation-url: <http://www.na-mic.org/>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This was extracted from the Hammer Registration source code, and wrapped up by Hans J. Johnson.

Inputs:

```
[Mandatory]

[Optional]
Scale: (an integer (int or long))
    Determine Scale of Ball
    flag: --Scale %d
Strength: (a float)
    Determine Strength of Edges
    flag: --Strength %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputCSFVolume: (an existing file name)
    Required: input CSF posterior image
    flag: --inputCSFVolume %s
inputGMVolume: (an existing file name)
    Required: input grey matter posterior image
    flag: --inputGMVolume %s
inputWMVolume: (an existing file name)
    Required: input white matter posterior image
    flag: --inputWMVolume %s
outputVolumeBase: (a unicode string)
    Required: output image base name to be appended for each feature
    vector.
    flag: --outputVolumeBase %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

None

NeighborhoodMean

[Link to code](#)

Wraps command **** NeighborhoodMean ****

title: Neighborhood Mean

category: Filtering.FeatureDetection

description: Calculates the mean, for the given neighborhood size, at each voxel of the T1, T2, and FLAIR.

version: 0.1.0.\$Revision: 1 \$(alpha)

documentation-url: <http://www.na-mic.org/>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    flag: --inputMaskVolume %s
inputRadius: (an integer (int or long))
    Required: input neighborhood radius
    flag: --inputRadius %d

```

```
inputVolume: (an existing file name)
    Required: input image
    flag: --inputVolume %s
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

NeighborhoodMedian

[Link to code](#)

Wraps command **** NeighborhoodMedian ****

title: Neighborhood Median

category: Filtering.FeatureDetection

description: Calculates the median, for the given neighborhood size, at each voxel of the input image.

version: 0.1.0.\$Revision: 1 \$(alpha)

documentation-url: <http://www.na-mic.org/>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    flag: --inputMaskVolume %s
inputRadius: (an integer (int or long))
    Required: input neighborhood radius
    flag: --inputRadius %d
inputVolume: (an existing file name)
    Required: input image
    flag: --inputVolume %s
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
```

```
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
Required: output image
```

STAPLEAnalysis[Link to code](#)Wraps command **** STAPLEAnalysis ****

title: Dilate Image

category: Filtering.FeatureDetection

description: Uses mathematical morphology to dilate the input images.

version: 0.1.0.\$Revision: 1 \$(alpha)

documentation-url: <http://www.na-mic.org/>license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Mark Scully and Jeremy Bockholt.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputDimension: (an integer (int or long))
    Required: input image Dimension 2 or 3
    flag: --inputDimension %d
inputLabelVolume: (a list of items which are an existing file name)
    Required: input label volume
    flag: --inputLabelVolume %s...
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
Required: output image
```

TextureFromNoiseImageFilter[Link to code](#)Wraps command **** TextureFromNoiseImageFilter ****

title: TextureFromNoiseImageFilter

category: Filtering.FeatureDetection
 description: Calculate the local noise in an image.
 version: 0.1.0.\$Revision: 1 \$(alpha)
 documentation-url: <http://www.na-mic.org/>
 license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>
 contributor: This tool was developed by Eunyoung Regina Kim
 Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputRadius: (an integer (int or long))
    Required: input neighborhood radius
    flag: --inputRadius %d
inputVolume: (an existing file name)
    Required: input image
    flag: --inputVolume %s
outputVolume: (a boolean or a file name)
    Required: output image
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

TextureMeasureFilter

[Link to code](#)

Wraps command `** TextureMeasureFilter **`

title: Canny Level Set Image Filter

category: Filtering.FeatureDetection

description: The CannySegmentationLevelSet is commonly used to refine a manually generated manual mask.

version: 0.3.0

license: CC

contributor: Regina Kim

acknowledgements: This command module was derived from Insight/Examples/Segmentation/CannySegmentationLevelSetImage (copyright) Insight Software Consortium. See http://wiki.na-mic.org/Wiki/index.php/Slicer3:Execution_Model_Documentation for more detailed descriptions.

Inputs:

```
[Mandatory]
```

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
distance: (an integer (int or long))
    flag: --distance %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputMaskVolume: (an existing file name)
    flag: --inputMaskVolume %s
inputVolume: (an existing file name)
    flag: --inputVolume %s
insideROIValue: (a float)
    flag: --insideROIValue %f
outputFilename: (a boolean or a file name)
    flag: --outputFilename %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputFilename: (an existing file name)
```

interfaces.semtools.legacy.registration

scalartransform

[Link to code](#)**Wraps command** `** scalartransform **`

title: ScalarTransform (DTIPProcess)

category: Legacy.Registration

version: 1.0.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/Nightly/Extensions/DTIPProcess>**license: Copyright (c) Casey Goodlett. All rights reserved.**

See <http://www.ia.unc.edu/dev/Copyright.htm> for details. This software is distributed WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the above copyright notices for more information.

contributor: Casey Goodlett

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
deformation: (an existing file name)
    Deformation field.
    flag: --deformation %s
```

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
h_field: (a boolean)
         The deformation is an h-field.
         flag: --h_field
ignore_exception: (a boolean, nipy default value: False)
         Print an error message instead of throwing an exception in case the
         interface fails to run
input_image: (an existing file name)
         Image to tranform
         flag: --input_image %s
interpolation: ('nearestneighbor' or 'linear' or 'cubic')
         Interpolation type (nearestneighbor, linear, cubic)
         flag: --interpolation %s
invert: (a boolean)
         Invert tranform before applying.
         flag: --invert
output_image: (a boolean or a file name)
         The transformed image
         flag: --output_image %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
         Control terminal output: `stream` - displays to terminal immediately
         (default), `allatonce` - waits till command is finished to display
         output, `file` - writes output to file, `none` - output is ignored
transformation: (a boolean or a file name)
         Output file for transformation parameters
         flag: --transformation %s

```

Outputs:

```

output_image: (an existing file name)
         The transformed image
transformation: (an existing file name)
         Output file for transformation parameters

```

interfaces.semtools.registration.brainsfit**BRAINSFit****Link to code**

Wraps command **** BRAINSFit ****

title: General Registration (BRAINS)

category: Registration

description: Register a three-dimensional volume to a reference volume (Mattes Mutual Information by default). Full documentation available here: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BRAINSFit>. Method described in BRAINSFit: Mutual Information Registrations of Whole-Brain 3D Images, Using the Insight Toolkit, Johnson H.J., Harris G., Williams K., The Insight Journal, 2007. <http://hdl.handle.net/1926/1291>

version: 3.0.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BRAINSFit>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Hans J. Johnson, hans-johnson -at- uiowa.edu, <http://www.psychiatry.uiowa.edu>

acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); Gregory Harris(1), Vincent Magnotta(1,2,3); Andriy Fedorov(5) 1=University of Iowa Department of Psychiatry, 2=University of Iowa Department of Radiol-

ogy, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering, 5=Surgical Planning Lab, Harvard

Inputs:

```
[Mandatory]

[Optional]
ROIAutoClosingSize: (a float)
    This flag is only relevant when using ROIAUTO mode for initializing
    masks. It defines the hole closing size in mm. It is rounded up to
    the nearest whole pixel size in each direction. The default is to
    use a closing size of 9mm. For mouse data this value may need to be
    reset to 0.9 or smaller.
    flag: --ROIAutoClosingSize %f
ROIAutoDilateSize: (a float)
    This flag is only relevant when using ROIAUTO mode for initializing
    masks. It defines the final dilation size to capture a bit of
    background outside the tissue region. A setting of 10mm has been
    shown to help regularize a BSpline registration type so that there
    is some background constraints to match the edges of the head
    better.
    flag: --ROIAutoDilateSize %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
backgroundFillValue: (a float)
    This value will be used for filling those areas of the output image
    that have no corresponding voxels in the input moving image.
    flag: --backgroundFillValue %f
bsplineTransform: (a boolean or a file name)
    (optional) Output estimated transform - in case the computed
    transform is BSpline. NOTE: You must set at least one output object
    (transform and/or output volume).
    flag: --bsplineTransform %s
costFunctionConvergenceFactor: (a float)
    From itkLBFGSBOptimizer.h: Set/Get the
    CostFunctionConvergenceFactor. Algorithm terminates when the
    reduction in cost function is less than (factor * epsmcj) where
    epsmcj is the machine precision. Typical values for factor: 1e+12
    for low accuracy; 1e+7 for moderate accuracy and 1e+1 for extremely
    high accuracy. 1e+9 seems to work well.,
    flag: --costFunctionConvergenceFactor %f
costMetric: ('MMI' or 'MSE' or 'NC' or 'MIH')
    The cost metric to be used during fitting. Defaults to MMI. Options
    are MMI (Mattes Mutual Information), MSE (Mean Square Error), NC
    (Normalized Correlation), MC (Match Cardinality for binary images)
    flag: --costMetric %s
debugLevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
    flag: --debugLevel %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
failureExitCode: (an integer (int or long))
    If the fit fails, exit with this status code. (It can be used to
    force a successful exit status of (0) if the registration fails due
```

```

    to reaching the maximum number of iterations.
    flag: --failureExitCode %d
fixedBinaryVolume: (an existing file name)
    Fixed Image binary mask volume, required if Masking Option is ROI.
    Image areas where the mask volume has zero value are ignored during
    the registration.
    flag: --fixedBinaryVolume %s
fixedVolume: (an existing file name)
    Input fixed image (the moving image will be transformed into this
    image space).
    flag: --fixedVolume %s
fixedVolume2: (an existing file name)
    Input fixed image that will be used for multimodal registration.
    (the moving image will be transformed into this image space).
    flag: --fixedVolume2 %s
fixedVolumeTimeIndex: (an integer (int or long))
    The index in the time series for the 3D fixed image to fit. Only
    allowed if the fixed input volume is 4-dimensional.
    flag: --fixedVolumeTimeIndex %d
gui: (a boolean)
    Display intermediate image volumes for debugging. NOTE: This is not
    part of the standard build sytem, and probably does nothing on your
    installation.
    flag: --gui
histogramMatch: (a boolean)
    Apply histogram matching operation for the input images to make them
    more similar. This is suitable for images of the same modality that
    may have different brightness or contrast, but the same overall
    intensity profile. Do NOT use if registering images from different
    modalities.
    flag: --histogramMatch
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initialTransform: (an existing file name)
    Transform to be applied to the moving image to initialize the
    registration. This can only be used if Initialize Transform Mode is
    Off.
    flag: --initialTransform %s
initializeRegistrationByCurrentGenericTransform: (a boolean)
    If this flag is ON, the current generic composite transform,
    resulted from the linear registration stages, is set to initialize
    the follow nonlinear registration process. However, by the default
    behaviour, the moving image is first warped based on the existant
    transform before it is passed to the BSpline registration filter. It
    is done to speed up the BSpline registration by reducing the
    computations of composite transform Jacobian.
    flag: --initializeRegistrationByCurrentGenericTransform
initializeTransformMode: ('Off' or 'useMomentsAlign' or
    'useCenterOfHeadAlign' or 'useGeometryAlign' or
    'useCenterOfROIAAlign')
    Determine how to initialize the transform center. useMomentsAlign
    assumes that the center of mass of the images represent similar
    structures. useCenterOfHeadAlign attempts to use the top of head and
    shape of neck to drive a center of mass estimate. useGeometryAlign
    on assumes that the center of the voxel lattice of the images
    represent similar structures. Off assumes that the physical space of
    the images are close. This flag is mutually exclusive with the

```

```

Initialization transform.
flag: --initializeTransformMode %s
interpolationMode: ('NearestNeighbor' or 'Linear' or
    'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
    'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
Type of interpolation to be used when applying transform to moving
volume. Options are Linear, NearestNeighbor, BSpline, WindowedSinc,
Hamming, Cosine, Welch, Lanczos, or ResampleInPlace. The
ResampleInPlace option will create an image with the same discrete
voxel values and will adjust the origin and direction of the
physical space interpretation.
flag: --interpolationMode %s
linearTransform: (a boolean or a file name)
    (optional) Output estimated transform - in case the computed
transform is not BSpline. NOTE: You must set at least one output
object (transform and/or output volume).
flag: --linearTransform %s
logFileReport: (a boolean or a file name)
    A file to write out final information report in CSV file: MetricName
    ,MetricValue,FixedImageName,FixedMaskName,MovingImageName,MovingMask
    Name
flag: --logFileReport %s
maskInferiorCutOffFromCenter: (a float)
    If Initialize Transform Mode is set to useCenterOfHeadAlign or
Masking Option is ROIAUTO then this value defines the how much is
cut of from the inferior part of the image. The cut-off distance is
specified in millimeters, relative to the image center. If the value
is 1000 or larger then no cut-off performed.
flag: --maskInferiorCutOffFromCenter %f
maskProcessingMode: ('NOMASK' or 'ROIAUTO' or 'ROI')
    Specifies a mask to only consider a certain image region for the
registration. If ROIAUTO is chosen, then the mask is computed using
Otsu thresholding and hole filling. If ROI is chosen then the mask
has to be specified as in input.
flag: --maskProcessingMode %s
maxBSplineDisplacement: (a float)
    Maximum allowed displacements in image physical coordinates (mm) for
BSpline control grid along each axis. A value of 0.0 indicates that
the problem should be unbounded. NOTE: This only constrains the
BSpline portion, and does not limit the displacement from the
associated bulk transform. This can lead to a substantial reduction
in computation time in the BSpline optimizer.,
flag: --maxBSplineDisplacement %f
maximumNumberOfCorrections: (an integer (int or long))
    Maximum number of corrections in lbfgsb optimizer.
flag: --maximumNumberOfCorrections %d
maximumNumberOfEvaluations: (an integer (int or long))
    Maximum number of evaluations for line search in lbfgsb optimizer.
flag: --maximumNumberOfEvaluations %d
maximumStepLength: (a float)
    Starting step length of the optimizer. In general, higher values
allow for recovering larger initial misalignments but there is an
increased chance that the registration will not converge.
flag: --maximumStepLength %f
medianFilterSize: (a list of items which are an integer (int or
    long))
    Apply median filtering to reduce noise in the input volumes. The 3
values specify the radius for the optional MedianImageFilter

```

```

preprocessing in all 3 directions (in voxels).
flag: --medianFilterSize %s
metricSamplingStrategy: ('Random')
    It defines the method that registration filter uses to sample the
    input fixed image. Only Random is supported for now.
    flag: --metricSamplingStrategy %s
minimumStepLength: (a list of items which are a float)
    Each step in the optimization takes steps at least this big. When
    none are possible, registration is complete. Smaller values allows
    the optimizer to make smaller adjustments, but the registration time
    may increase.
    flag: --minimumStepLength %s
movingBinaryVolume: (an existing file name)
    Moving Image binary mask volume, required if Masking Option is ROI.
    Image areas where the mask volume has zero value are ignored during
    the registration.
    flag: --movingBinaryVolume %s
movingVolume: (an existing file name)
    Input moving image (this image will be transformed into the fixed
    image space).
    flag: --movingVolume %s
movingVolume2: (an existing file name)
    Input moving image that will be used for multimodal
    registration(this image will be transformed into the fixed image
    space).
    flag: --movingVolume2 %s
movingVolumeTimeIndex: (an integer (int or long))
    The index in the time series for the 3D moving image to fit. Only
    allowed if the moving input volume is 4-dimensional
    flag: --movingVolumeTimeIndex %d
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels used for mutual information metric
    estimation.
    flag: --numberOfHistogramBins %d
numberOfIterations: (a list of items which are an integer (int or
    long))
    The maximum number of iterations to try before stopping the
    optimization. When using a lower value (500-1000) then the
    registration is forced to terminate earlier but there is a higher
    risk of stopping before an optimal solution is reached.
    flag: --numberOfIterations %s
numberOfMatchPoints: (an integer (int or long))
    Number of histogram match points used for mutual information metric
    estimation.
    flag: --numberOfMatchPoints %d
numberOfSamples: (an integer (int or long))
    The number of voxels sampled for mutual information computation.
    Increase this for higher accuracy, at the cost of longer computation
    time., NOTE that it is suggested to use samplingPercentage instead
    of this option. However, if set to non-zero, numberOfSamples
    overwrites the samplingPercentage option.
    flag: --numberOfSamples %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use. (default is
    auto-detected)
    flag: --numberOfThreads %d
outputFixedVolumeROI: (a boolean or a file name)
    ROI that is automatically computed from the fixed image. Only

```

```

    available if Masking Option is ROIAUTO. Image areas where the mask
    volume has zero value are ignored during the registration.
    flag: --outputFixedVolumeROI %s
outputMovingVolumeROI: (a boolean or a file name)
    ROI that is automatically computed from the moving image. Only
    available if Masking Option is ROIAUTO. Image areas where the mask
    volume has zero value are ignored during the registration.
    flag: --outputMovingVolumeROI %s
outputTransform: (a boolean or a file name)
    (optional) Filename to which save the (optional) estimated
    transform. NOTE: You must select either the outputTransform or the
    outputVolume option.
    flag: --outputTransform %s
outputVolume: (a boolean or a file name)
    (optional) Output image: the moving image warped to the fixed image
    space. NOTE: You must set at least one output object (transform
    and/or output volume).
    flag: --outputVolume %s
outputVolumePixelType: ('float' or 'short' or 'ushort' or 'int' or
    'uint' or 'uchar')
    Data type for representing a voxel of the Output Volume.
    flag: --outputVolumePixelType %s
projectedGradientTolerance: (a float)
    From itkLBFGSBOptimizer.h: Set/Get the ProjectedGradientTolerance.
    Algorithm terminates when the project gradient is below the
    tolerance. Default lbfgsb value is 1e-5, but 1e-4 seems to work
    well.,
    flag: --projectedGradientTolerance %f
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    flag: --promptUser
relaxationFactor: (a float)
    Specifies how quickly the optimization step length is decreased
    during registration. The value must be larger than 0 and smaller
    than 1. Larger values result in slower step size decrease, which
    allow for recovering larger initial misalignments but it increases
    the registration time and the chance that the registration will not
    converge.
    flag: --relaxationFactor %f
removeIntensityOutliers: (a float)
    Remove very high and very low intensity voxels from the input
    volumes. The parameter specifies the half percentage to decide
    outliers of image intensities. The default value is zero, which
    means no outlier removal. If the value of 0.005 is given, the 0.005%
    of both tails will be thrown away, so 0.01% of intensities in total
    would be ignored in the statistic calculation.
    flag: --removeIntensityOutliers %f
reproportionScale: (a float)
    ScaleVersor3D 'Scale' compensation factor. Increase this to allow
    for more rescaling in a ScaleVersor3D or ScaleSkewVersor3D search
    pattern. 1.0 works well with a translationScale of 1000.0
    flag: --reproportionScale %f
samplingPercentage: (a float)
    Fraction of voxels of the fixed image that will be used for
    registration. The number has to be larger than zero and less or
    equal to one. Higher values increase the computation time but may
    give more accurate results. You can also limit the sampling focus

```

```
with ROI masks and ROIAUTO mask generation. The default is 0.002
(use approximately 0.2% of voxels, resulting in 100000 samples in a
512x512x192 volume) to provide a very fast registration in most
cases. Typical values range from 0.01 (1%) for low detail images to
0.2 (20%) for high detail images.
flag: --samplingPercentage %f
scaleOutputValues: (a boolean)
    If true, and the voxel values do not fit within the minimum and
    maximum values of the desired outputVolumePixelType, then linearly
    scale the min/max output image voxel values to fit within the
    min/max range of the outputVolumePixelType.
    flag: --scaleOutputValues
skewScale: (a float)
    ScaleSkewVersor3D Skew compensation factor. Increase this to allow
    for more skew in a ScaleSkewVersor3D search pattern. 1.0 works well
    with a translationScale of 1000.0
    flag: --skewScale %f
splineGridSize: (a list of items which are an integer (int or long))
    Number of BSpline grid subdivisions along each axis of the fixed
    image, centered on the image space. Values must be 3 or higher for
    the BSpline to be correctly computed.
    flag: --splineGridSize %s
strippedOutputTransform: (a boolean or a file name)
    Rigid component of the estimated affine transform. Can be used to
    rigidly register the moving image to the fixed image. NOTE: This
    value is overridden if either bsplineTransform or linearTransform is
    set.
    flag: --strippedOutputTransform %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformType: (a list of items which are a unicode string)
    Specifies a list of registration types to be used. The valid types
    are, Rigid, ScaleVersor3D, ScaleSkewVersor3D, Affine, BSpline and
    SyN. Specifying more than one in a comma separated list will
    initialize the next stage with the previous results. If
    registrationClass flag is used, it overrides this parameter setting.
    flag: --transformType %s
translationScale: (a float)
    How much to scale up changes in position (in mm) compared to unit
    rotational changes (in radians) -- decrease this to allow for more
    rotation in the search pattern.
    flag: --translationScale %f
useAffine: (a boolean)
    Perform an Affine registration as part of the sequential
    registration steps. This family of options overrides the use of
    transformType if any of them are set.
    flag: --useAffine
useBSpline: (a boolean)
    Perform a BSpline registration as part of the sequential
    registration steps. This family of options overrides the use of
    transformType if any of them are set.
    flag: --useBSpline
useComposite: (a boolean)
    Perform a Composite registration as part of the sequential
    registration steps. This family of options overrides the use of
    transformType if any of them are set.
```

```

    flag: --useComposite
useROIBSpline: (a boolean)
    If enabled then the bounding box of the input ROIs defines the
    BSpline grid support region. Otherwise the BSpline grid support
    region is the whole fixed image.
    flag: --useROIBSpline
useRigid: (a boolean)
    Perform a rigid registration as part of the sequential registration
    steps. This family of options overrides the use of transformType if
    any of them are set.
    flag: --useRigid
useScaleSkewVersor3D: (a boolean)
    Perform a ScaleSkewVersor3D registration as part of the sequential
    registration steps. This family of options overrides the use of
    transformType if any of them are set.
    flag: --useScaleSkewVersor3D
useScaleVersor3D: (a boolean)
    Perform a ScaleVersor3D registration as part of the sequential
    registration steps. This family of options overrides the use of
    transformType if any of them are set.
    flag: --useScaleVersor3D
useSyN: (a boolean)
    Perform a SyN registration as part of the sequential registration
    steps. This family of options overrides the use of transformType if
    any of them are set.
    flag: --useSyN
writeOutputTransformInFloat: (a boolean)
    By default, the output registration transforms (either the output
    composite transform or each transform component) are written to the
    disk in double precision. If this flag is ON, the output transforms
    will be written in single (float) precision. It is especially
    important if the output transform is a displacement field transform,
    or it is a composite transform that includes several displacement
    fields.
    flag: --writeOutputTransformInFloat
writeTransformOnFailure: (a boolean)
    Flag to save the final transform even if the numberOfIterations are
    reached without convergence. (Intended for use when
    --failureExitCode 0 )
    flag: --writeTransformOnFailure

```

Outputs:

```

bsplineTransform: (an existing file name)
    (optional) Output estimated transform - in case the computed
    transform is BSpline. NOTE: You must set at least one output object
    (transform and/or output volume).
linearTransform: (an existing file name)
    (optional) Output estimated transform - in case the computed
    transform is not BSpline. NOTE: You must set at least one output
    object (transform and/or output volume).
logFileReport: (an existing file name)
    A file to write out final information report in CSV file: MetricName
    ,MetricValue,FixedImageName,FixedMaskName,MovingImageName,MovingMask
    Name
outputFixedVolumeROI: (an existing file name)
    ROI that is automatically computed from the fixed image. Only
    available if Masking Option is ROIAUTO. Image areas where the mask

```

```
volume has zero value are ignored during the registration.
outputMovingVolumeROI: (an existing file name)
    ROI that is automatically computed from the moving image. Only
    available if Masking Option is ROIAUTO. Image areas where the mask
    volume has zero value are ignored during the registration.
outputTransform: (an existing file name)
    (optional) Filename to which save the (optional) estimated
    transform. NOTE: You must select either the outputTransform or the
    outputVolume option.
outputVolume: (an existing file name)
    (optional) Output image: the moving image warped to the fixed image
    space. NOTE: You must set at least one output object (transform
    and/or output volume).
strippedOutputTransform: (an existing file name)
    Rigid component of the estimated affine transform. Can be used to
    rigidly register the moving image to the fixed image. NOTE: This
    value is overridden if either bsplineTransform or linearTransform is
    set.
```

interfaces.semtools.registration.brainsresample

BRAINSResample

[Link to code](#)

Wraps command **** BRAINSResample ****

title: Resample Image (BRAINS)

category: Registration

description: This program collects together three common image processing tasks that all involve resampling an image volume: Resampling to a new resolution and spacing, applying a transformation (using an ITK transform IO mechanisms) and Warping (using a vector image deformation field). Full documentation available here: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BRAINSResample>.

version: 3.0.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BRAINSResample>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Vincent Magnotta, Greg Harris, and Hans Johnson.

acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
defaultValue: (a float)
    Default voxel value
    flag: --defaultValue %f
deformationVolume: (an existing file name)
    Displacement Field to be used to warp the image (ITKv3 or earlier)
    flag: --deformationVolume %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
```



```

    of class 'str', nipy default value: {})
    Environment variables
gridSpacing: (a list of items which are an integer (int or long))
    Add warped grid to output image to help show the deformation that
    occurred with specified spacing. A spacing of 0 in a dimension
    indicates that grid lines should be rendered to fall exactly (i.e.
    do not allow displacements off that plane). This is useful for
    making a 2D image of grid lines from the 3D space
    flag: --gridSpacing %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Image To Warp
    flag: --inputVolume %s
interpolationMode: ('NearestNeighbor' or 'Linear' or
    'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
    'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
    Type of interpolation to be used when applying transform to moving
    volume. Options are Linear, ResampleInPlace, NearestNeighbor,
    BSpline, or WindowedSinc
    flag: --interpolationMode %s
inverseTransform: (a boolean)
    True/False is to compute inverse of given transformation. Default is
    false
    flag: --inverseTransform
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Resulting deformed image
    flag: --outputVolume %s
pixelType: ('float' or 'short' or 'ushort' or 'int' or 'uint' or
    'uchar' or 'binary')
    Specifies the pixel type for the input/output images. The 'binary'
    pixel type uses a modified algorithm whereby the image is read in as
    unsigned char, a signed distance map is created, signed distance map
    is resampled, and then a thresholded image of type unsigned char is
    written to disk.
    flag: --pixelType %s
referenceVolume: (an existing file name)
    Reference image used only to define the output space. If not
    specified, the warping is done in the same space as the image to
    warp.
    flag: --referenceVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
warpTransform: (an existing file name)
    Filename for the BRAINSFit transform (ITKv3 or earlier) or composite
    transform file (ITKv4)
    flag: --warpTransform %s

```

Outputs:

```

outputVolume: (an existing file name)
    Resulting deformed image

```

interfaces.semtools.registration.brainsresize

BRAINSResize

[Link to code](#)

Wraps command **** BRAINSResize ****

title: Resize Image (BRAINS)

category: Registration

description: This program is useful for downsampling an image by a constant scale factor.

version: 3.0.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Hans Johnson.

acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Image To Scale
    flag: --inputVolume %s
outputVolume: (a boolean or a file name)
    Resulting scaled image
    flag: --outputVolume %s
pixelType: ('float' or 'short' or 'ushort' or 'int' or 'uint' or
    'uchar' or 'binary')
    Specifies the pixel type for the input/output images. The 'binary'
    pixel type uses a modified algorithm whereby the image is read in as
    unsigned char, a signed distance map is created, signed distance map
    is resampled, and then a thresholded image of type unsigned char is
    written to disk.
    flag: --pixelType %s
scaleFactor: (a float)
    The scale factor for the image spacing.
    flag: --scaleFactor %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Resulting scaled image

```

interfaces.semtools.registration.specialized

BRAINSDemonWarp

[Link to code](#)

Wraps command **** BRAINSDemonWarp ****

title: Demon Registration (BRAINS)

category: Registration.Specialized

description: This program finds a deformation field to warp a moving image onto a fixed image. The images must be of the same signal kind, and contain an image of the same kind of object. This program uses the Thirion Demons warp software in ITK, the Insight Toolkit. Additional information is available at: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BRAINSDemonWarp>.

version: 3.0.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BRAINSDemonWarp>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Hans J. Johnson and Greg Harris.

acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
arrayOfPyramidLevelIterations: (a list of items which are an integer
    (int or long))
    The number of iterations for each pyramid level
    flag: --arrayOfPyramidLevelIterations %s
backgroundFillValue: (an integer (int or long))
    Replacement value to overwrite background when performing BOBF
    flag: --backgroundFillValue %d
checkerboardPatternSubdivisions: (a list of items which are an
    integer (int or long))
    Number of Checkerboard subdivisions in all 3 directions
    flag: --checkerboardPatternSubdivisions %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixedBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Fixed image.
    flag: --fixedBinaryVolume %s
fixedVolume: (an existing file name)
    Required: input fixed (target) image
    flag: --fixedVolume %s
gradient_type: ('0' or '1' or '2')
    Type of gradient used for computing the demons force (0 is
    symmetrized, 1 is fixed image, 2 is moving image)
    flag: --gradient_type %s
gui: (a boolean)
    Display intermediate image volumes for debugging
    flag: --gui
histogramMatch: (a boolean)
```

```

Histogram Match the input images. This is suitable for images of the
same modality that may have different absolute scales, but the same
overall intensity profile.
flag: --histogramMatch
ignore_exception: (a boolean, nipy default value: False)
Print an error message instead of throwing an exception in case the
interface fails to run
initializeWithDisplacementField: (an existing file name)
Initial deformation field vector image file name
flag: --initializeWithDisplacementField %s
initializeWithTransform: (an existing file name)
Initial Transform filename
flag: --initializeWithTransform %s
inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
Input volumes will be typecast to this format:
float|short|ushort|int|uchar
flag: --inputPixelType %s
interpolationMode: ('NearestNeighbor' or 'Linear' or
'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
Type of interpolation to be used when applying transform to moving
volume. Options are Linear, ResampleInPlace, NearestNeighbor,
BSpline, or WindowedSinc
flag: --interpolationMode %s
lowerThresholdForBOBF: (an integer (int or long))
Lower threshold for performing BOBF
flag: --lowerThresholdForBOBF %d
maskProcessingMode: ('NOMASK' or 'ROIAUTO' or 'ROI' or 'BOBF')
What mode to use for using the masks: NOMASK|ROIAUTO|ROI|BOBF. If
ROIAUTO is choosen, then the mask is implicitly defined using a otsu
foreground and hole filling algorithm. Where the Region Of Interest
mode uses the masks to define what parts of the image should be used
for computing the deformation field. Brain Only Background Fill uses
the masks to pre-process the input images by clipping and filling in
the background with a predefined value.
flag: --maskProcessingMode %s
max_step_length: (a float)
Maximum length of an update vector (0: no restriction)
flag: --max_step_length %f
medianFilterSize: (a list of items which are an integer (int or
long))
Median filter radius in all 3 directions. When images have a lot of
salt and pepper noise, this step can improve the registration.
flag: --medianFilterSize %s
minimumFixedPyramid: (a list of items which are an integer (int or
long))
The shrink factor for the first level of the fixed image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
full scale)
flag: --minimumFixedPyramid %s
minimumMovingPyramid: (a list of items which are an integer (int or
long))
The shrink factor for the first level of the moving image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
full scale)
flag: --minimumMovingPyramid %s
movingBinaryVolume: (an existing file name)
Mask filename for desired region of interest in the Moving image.

```

```

    flag: --movingBinaryVolume %s
movingVolume: (an existing file name)
    Required: input moving image
    flag: --movingVolume %s
neighborhoodForBOBF: (a list of items which are an integer (int or
    long))
    neighborhood in all 3 directions to be included when performing BOBF
    flag: --neighborhoodForBOBF %s
numberOfBCHApproximationTerms: (an integer (int or long))
    Number of terms in the BCH expansion
    flag: --numberOfBCHApproximationTerms %d
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels
    flag: --numberOfHistogramBins %d
numberOfMatchPoints: (an integer (int or long))
    The number of match points for histogramMatch
    flag: --numberOfMatchPoints %d
numberOfPyramidLevels: (an integer (int or long))
    Number of image pyramid levels to use in the multi-resolution
    registration.
    flag: --numberOfPyramidLevels %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputCheckerboardVolume: (a boolean or a file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
    flag: --outputCheckerboardVolume %s
outputDebug: (a boolean)
    Flag to write debugging images after each step.
    flag: --outputDebug
outputDisplacementFieldPrefix: (a unicode string)
    Displacement field filename prefix for writing separate x, y, and z
    component images
    flag: --outputDisplacementFieldPrefix %s
outputDisplacementFieldVolume: (a boolean or a file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
    flag: --outputDisplacementFieldVolume %s
outputNormalized: (a boolean)
    Flag to warp and write the normalized images to output. In
    normalized images the image values are fit-scaled to be between 0
    and the maximum storage type value.
    flag: --outputNormalized
outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    outputVolume will be typecast to this format:
    float|short|ushort|int|uchar
    flag: --outputPixelType %s
outputVolume: (a boolean or a file name)
    Required: output resampled moving image (will have the same physical
    space as the fixedVolume).
    flag: --outputVolume %s
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    flag: --promptUser
registrationFilterType: ('Demons' or 'FastSymmetricForces' or
    'Diffeomorphic')

```

```
Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic
flag: --registrationFilterType %s
seedForBOBF: (a list of items which are an integer (int or long))
    coordinates in all 3 directions for Seed when performing BOBF
flag: --seedForBOBF %s
smoothDisplacementFieldSigma: (a float)
    A gaussian smoothing value to be applied to the deformation feild at
    each iteration.
flag: --smoothDisplacementFieldSigma %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
upFieldSmoothing: (a float)
    Smoothing sigma for the update field at each iteration
flag: --upFieldSmoothing %f
upperThresholdForBOBF: (an integer (int or long))
    Upper threshold for performing BOBF
flag: --upperThresholdForBOBF %d
use_vanilla_dem: (a boolean)
    Run vanilla demons algorithm
flag: --use_vanilla_dem
```

Outputs:

```
outputCheckerboardVolume: (an existing file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
outputDisplacementFieldVolume: (an existing file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
outputVolume: (an existing file name)
    Required: output resampled moving image (will have the same physical
    space as the fixedVolume).
```

BRAINSTransformFromFiducials

[Link to code](#)

Wraps command **** BRAINSTransformFromFiducials ****

title: Fiducial Registration (BRAINS)

category: Registration.Specialized

description: Computes a rigid, similarity or affine transform from a matched list of fiducials

version: 0.1.0.\$Revision\$

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Modules:TransformFromFiducials-Documentation-3.6>

contributor: Casey B Goodlett

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
```

```

    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
Environment variables
fixedLandmarks: (a list of items which are a list of from 3 to 3
    items which are a float)
    Ordered list of landmarks in the fixed image
    flag: --fixedLandmarks %s...
fixedLandmarksFile: (an existing file name)
    An fcsv formatted file with a list of landmark points.
    flag: --fixedLandmarksFile %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
movingLandmarks: (a list of items which are a list of from 3 to 3
    items which are a float)
    Ordered list of landmarks in the moving image
    flag: --movingLandmarks %s...
movingLandmarksFile: (an existing file name)
    An fcsv formatted file with a list of landmark points.
    flag: --movingLandmarksFile %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
saveTransform: (a boolean or a file name)
    Save the transform that results from registration
    flag: --saveTransform %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformType: ('Translation' or 'Rigid' or 'Similarity')
    Type of transform to produce
    flag: --transformType %s

```

Outputs:

```

saveTransform: (an existing file name)
    Save the transform that results from registration

```

VBRAINSDemonWarp**Link to code**

Wraps command **** VBRAINSDemonWarp ****

title: Vector Demon Registration (BRAINS)

category: Registration.Specialized

description: This program finds a deformation field to warp a moving image onto a fixed image. The images must be of the same signal kind, and contain an image of the same kind of object. This program uses the Thirion Demons warp software in ITK, the Insight Toolkit. Additional information is available at: <http://www.nitrc.org/projects/brainsdemonwarp>.

version: 3.0.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/VBRAINSDemonWarp>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Hans J. Johnson and Greg Harris.

acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
arrayOfPyramidLevelIterations: (a list of items which are an integer
    (int or long))
    The number of iterations for each pyramid level
    flag: --arrayOfPyramidLevelIterations %s
backgroundFillValue: (an integer (int or long))
    Replacement value to overwrite background when performing BOBF
    flag: --backgroundFillValue %d
checkerboardPatternSubdivisions: (a list of items which are an
    integer (int or long))
    Number of Checkerboard subdivisions in all 3 directions
    flag: --checkerboardPatternSubdivisions %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
fixedBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Fixed image.
    flag: --fixedBinaryVolume %s
fixedVolume: (a list of items which are an existing file name)
    Required: input fixed (target) image
    flag: --fixedVolume %s...
gradient_type: ('0' or '1' or '2')
    Type of gradient used for computing the demons force (0 is
    symmetrized, 1 is fixed image, 2 is moving image)
    flag: --gradient_type %s
gui: (a boolean)
    Display intermediate image volumes for debugging
    flag: --gui
histogramMatch: (a boolean)
    Histogram Match the input images. This is suitable for images of the
    same modality that may have different absolute scales, but the same
    overall intensity profile.
    flag: --histogramMatch
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initializeWithDisplacementField: (an existing file name)
    Initial deformation field vector image file name
    flag: --initializeWithDisplacementField %s
initializeWithTransform: (an existing file name)
    Initial Transform filename
    flag: --initializeWithTransform %s
inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    Input volumes will be typecast to this format:
    float|short|ushort|int|uchar
    flag: --inputPixelType %s
interpolationMode: ('NearestNeighbor' or 'Linear' or
    'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
    'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
    Type of interpolation to be used when applying transform to moving
    volume. Options are Linear, ResampleInPlace, NearestNeighbor,
```



```

BSpline, or WindowedSinc
flag: --interpolationMode %s
lowerThresholdForBOBF: (an integer (int or long))
    Lower threshold for performing BOBF
    flag: --lowerThresholdForBOBF %d
makeBOBF: (a boolean)
    Flag to make Brain-Only Background-Filled versions of the input and
    target volumes.
    flag: --makeBOBF
max_step_length: (a float)
    Maximum length of an update vector (0: no restriction)
    flag: --max_step_length %f
medianFilterSize: (a list of items which are an integer (int or
    long))
    Median filter radius in all 3 directions. When images have a lot of
    salt and pepper noise, this step can improve the registration.
    flag: --medianFilterSize %s
minimumFixedPyramid: (a list of items which are an integer (int or
    long))
    The shrink factor for the first level of the fixed image pyramid.
    (i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
    full scale)
    flag: --minimumFixedPyramid %s
minimumMovingPyramid: (a list of items which are an integer (int or
    long))
    The shrink factor for the first level of the moving image pyramid.
    (i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
    full scale)
    flag: --minimumMovingPyramid %s
movingBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Moving image.
    flag: --movingBinaryVolume %s
movingVolume: (a list of items which are an existing file name)
    Required: input moving image
    flag: --movingVolume %s...
neighborhoodForBOBF: (a list of items which are an integer (int or
    long))
    neighborhood in all 3 directions to be included when performing BOBF
    flag: --neighborhoodForBOBF %s
numberOfBCHApproximationTerms: (an integer (int or long))
    Number of terms in the BCH expansion
    flag: --numberOfBCHApproximationTerms %d
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels
    flag: --numberOfHistogramBins %d
numberOfMatchPoints: (an integer (int or long))
    The number of match points for histogramMatch
    flag: --numberOfMatchPoints %d
numberOfPyramidLevels: (an integer (int or long))
    Number of image pyramid levels to use in the multi-resolution
    registration.
    flag: --numberOfPyramidLevels %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputCheckerboardVolume: (a boolean or a file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.

```

```

    flag: --outputCheckerboardVolume %s
outputDebug: (a boolean)
    Flag to write debugging images after each step.
    flag: --outputDebug
outputDisplacementFieldPrefix: (a unicode string)
    Displacement field filename prefix for writing separate x, y, and z
    component images
    flag: --outputDisplacementFieldPrefix %s
outputDisplacementFieldVolume: (a boolean or a file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
    flag: --outputDisplacementFieldVolume %s
outputNormalized: (a boolean)
    Flag to warp and write the normalized images to output. In
    normalized images the image values are fit-scaled to be between 0
    and the maximum storage type value.
    flag: --outputNormalized
outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    outputVolume will be typecast to this format:
    float|short|ushort|int|uchar
    flag: --outputPixelType %s
outputVolume: (a boolean or a file name)
    Required: output resampled moving image (will have the same physical
    space as the fixedVolume).
    flag: --outputVolume %s
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    flag: --promptUser
registrationFilterType: ('Demons' or 'FastSymmetricForces' or
    'Diffeomorphic' or 'LogDemons' or 'SymmetricLogDemons')
    Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic|L
    ogDemons|SymmetricLogDemons
    flag: --registrationFilterType %s
seedForBOBF: (a list of items which are an integer (int or long))
    coordinates in all 3 directions for Seed when performing BOBF
    flag: --seedForBOBF %s
smoothDisplacementFieldSigma: (a float)
    A gaussian smoothing value to be applied to the deformation feild at
    each iteration.
    flag: --smoothDisplacementFieldSigma %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
upFieldSmoothing: (a float)
    Smoothing sigma for the update field at each iteration
    flag: --upFieldSmoothing %f
upperThresholdForBOBF: (an integer (int or long))
    Upper threshold for performing BOBF
    flag: --upperThresholdForBOBF %d
use_vanilla_dem: (a boolean)
    Run vanilla demons algorithm
    flag: --use_vanilla_dem
weightFactors: (a list of items which are a float)
    Weight fatctors for each input images
    flag: --weightFactors %s

```

Outputs:

```

outputCheckerboardVolume: (an existing file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
outputDisplacementFieldVolume: (an existing file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
outputVolume: (an existing file name)
    Required: output resampled moving image (will have the same physical
    space as the fixedVolume).

```

interfaces.semtools.segmentation.specialized

BRAINSABC

[Link to code](#)

Wraps command **** BRAINSABC ****

title: Intra-subject registration, bias Correction, and tissue classification (BRAINS)

category: Segmentation.Specialized

description: Atlas-based tissue segmentation method. This is an algorithmic extension of work done by XXXX at UNC and Utah XXXX need more description here.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
atlasDefinition: (an existing file name)
    Contains all parameters for Atlas
    flag: --atlasDefinition %s
atlasToSubjectInitialTransform: (a boolean or a file name)
    The initial transform from atlas to the subject
    flag: --atlasToSubjectInitialTransform %s
atlasToSubjectTransform: (a boolean or a file name)
    The transform from atlas to the subject
    flag: --atlasToSubjectTransform %s
atlasToSubjectTransformType: ('Identity' or 'Rigid' or 'Affine' or
    'BSpline' or 'SyN')
    What type of linear transform type do you want to use to register
    the atlas to the reference subject image.
    flag: --atlasToSubjectTransformType %s
atlasWarpingOff: (a boolean)
    Deformable registration of atlas to subject
    flag: --atlasWarpingOff
debuglevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
    flag: --debuglevel %d
defaultSuffix: (a unicode string)
    flag: --defaultSuffix %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
filterIteration: (an integer (int or long))

```

```
    Filter iterations
    flag: --filterIteration %d
filterMethod: ('None' or 'CurvatureFlow' or
    'GradientAnisotropicDiffusion' or 'Median')
    Filter method for preprocessing of registration
    flag: --filterMethod %s
filterTimeStep: (a float)
    Filter time step should be less than (PixelSpacing/(1^(DIM+1))),
    value is set to negative, then allow automatic setting of this
    value.
    flag: --filterTimeStep %f
gridSize: (a list of items which are an integer (int or long))
    Grid size for atlas warping with BSplines
    flag: --gridSize %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
implicitOutputs: (a boolean or a list of items which are a file name)
    Outputs to be made available to NiType. Needed because not all
    BRAINSABC outputs have command line arguments.
    flag: --implicitOutputs %s...
inputVolumeTypes: (a list of items which are a unicode string)
    The list of input image types corresponding to the inputVolumes.
    flag: --inputVolumeTypes %s
inputVolumes: (a list of items which are an existing file name)
    The list of input image files to be segmented.
    flag: --inputVolumes %s...
interpolationMode: ('BSpline' or 'NearestNeighbor' or 'WindowedSinc'
    or 'Linear' or 'ResampleInPlace' or 'Hamming' or 'Cosine' or
    'Welch' or 'Lanczos' or 'Blackman')
    Type of interpolation to be used when applying transform to moving
    volume. Options are Linear, NearestNeighbor, BSpline, WindowedSinc,
    or ResampleInPlace. The ResampleInPlace option will create an image
    with the same discrete voxel values and will adjust the origin and
    direction of the physical space interpretation.
    flag: --interpolationMode %s
maxBiasDegree: (an integer (int or long))
    Maximum bias degree
    flag: --maxBiasDegree %d
maxIterations: (an integer (int or long))
    Filter iterations
    flag: --maxIterations %d
medianFilterSize: (a list of items which are an integer (int or
    long))
    The radius for the optional MedianImageFilter preprocessing in all 3
    directions.
    flag: --medianFilterSize %s
numberOfSubSamplesInEachPlugArea: (a list of items which are an
    integer (int or long))
    Number of continous index samples taken at each direction of lattice
    space for each plug volume.
    flag: --numberOfSubSamplesInEachPlugArea %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputDir: (a boolean or a directory name)
    Ouput directory
    flag: --outputDir %s
```

```

outputDirtyLabels: (a boolean or a file name)
    Output Dirty Label Image
    flag: --outputDirtyLabels %s
outputFormat: ('NIFTI' or 'Meta' or 'Nrrd')
    Output format
    flag: --outputFormat %s
outputLabels: (a boolean or a file name)
    Output Label Image
    flag: --outputLabels %s
outputVolumes: (a boolean or a list of items which are a file name)
    Corrected Output Images: should specify the same number of images as
    inputVolume, if only one element is given, then it is used as a file
    pattern where %s is replaced by the imageVolumeType, and %d by the
    index list location.
    flag: --outputVolumes %s...
posteriorTemplate: (a unicode string)
    filename template for Posterior output files
    flag: --posteriorTemplate %s
purePlugsThreshold: (a float)
    If this threshold value is greater than zero, only pure samples are
    used to compute the distributions in EM classification, and only
    pure samples are used for KNN training. The default value is set to
    0, that means not using pure plugs. However, a value of 0.2 is
    suggested if you want to activate using pure plugs option.
    flag: --purePlugsThreshold %f
restoreState: (an existing file name)
    The initial state for the registration process
    flag: --restoreState %s
saveState: (a boolean or a file name)
    (optional) Filename to which save the final state of the
    registration
    flag: --saveState %s
subjectIntermodeTransformType: ('Identity' or 'Rigid' or 'Affine' or
    'BSpline')
    What type of linear transform type do you want to use to register
    the atlas to the reference subject image.
    flag: --subjectIntermodeTransformType %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
useKNN: (a boolean)
    Use the KNN stage of estimating posteriors.
    flag: --useKNN
writeLess: (a boolean)
    Does not write posteriors and filtered, bias corrected images
    flag: --writeLess

```

Outputs:

```

atlasToSubjectInitialTransform: (an existing file name)
    The initial transform from atlas to the subject
atlasToSubjectTransform: (an existing file name)
    The transform from atlas to the subject
implicitOutputs: (a list of items which are an existing file name)
    Outputs to be made available to NiType. Needed because not all
    BRAINSABC outputs have command line arguments.
outputDir: (an existing directory name)

```

```

    Ouput directory
outputDirtyLabels: (an existing file name)
    Output Dirty Label Image
outputLabels: (an existing file name)
    Output Label Image
outputVolumes: (a list of items which are an existing file name)
    Corrected Output Images: should specify the same number of images as
    inputVolume, if only one element is given, then it is used as a file
    pattern where %s is replaced by the imageVolumeType, and %d by the
    index list location.
saveState: (an existing file name)
    (optional) Filename to which save the final state of the
    registration

```

BRAINSConstellationDetector

[Link to code](#)

Wraps command **** BRAINSConstellationDetector ****

title: Brain Landmark Constellation Detector (BRAINS)

category: Segmentation.Specialized

description: This program will find the mid-sagittal plane, a constellation of landmarks in a volume, and create an AC/PC aligned data set with the AC point at the center of the voxel lattice (labeled at the origin of the image physical space.) Part of this work is an extention of the algorithms originally described by Dr. Babak A. Ardekani, Alvin H. Bachman, Model-based automatic detection of the anterior and posterior commissures on MRI scans, NeuroImage, Volume 46, Issue 3, 1 July 2009, Pages 677-682, ISSN 1053-8119, DOI: 10.1016/j.neuroimage.2009.02.030. (<http://www.sciencedirect.com/science/article/B6WNP-4VRP25C-4/2/8207b962a38aa83c822c6379bc43fe4c>)

version: 1.0

documentation-url: <http://www.nitrc.org/projects/brainscdetector/>

Inputs:

```

[Mandatory]

[Optional]
BackgroundFillValue: (a unicode string)
    Fill the background of image with specified short int value. Enter
    number or use BIGNEG for a large negative number.
    flag: --BackgroundFillValue %s
LLSModel: (an existing file name)
    Linear least squares model filename in HD5 format
    flag: --LLSModel %s
acLowerBound: (a float)
    , When generating a resampled output image, replace the image with
    the BackgroundFillValue everywhere below the plane This Far in
    physical units (millimeters) below (inferior to) the AC point (as
    found by the model.) The oversize default was chosen to have no
    effect. Based on visualizing a thousand masks in the IPIG study, we
    recommend a limit no smaller than 80.0 mm.,
    flag: --acLowerBound %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
atlasLandmarkWeights: (an existing file name)
    Weights associated with atlas landmarks to be used for BRAINSFit
    registration initialization,
    flag: --atlasLandmarkWeights %s

```

```

atlasLandmarks: (an existing file name)
    Atlas landmarks to be used for BRAINSFit registration
    initialization,
    flag: --atlasLandmarks %s
atlasVolume: (an existing file name)
    Atlas volume image to be used for BRAINSFit registration
    flag: --atlasVolume %s
cutOutHeadInOutputVolume: (a boolean)
    , Flag to cut out just the head tissue when producing an
    (un)transformed clipped volume.,
    flag: --cutOutHeadInOutputVolume
debug: (a boolean)
    , Show internal debugging information.,
    flag: --debug
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
forceACPoint: (a list of items which are a float)
    , Use this flag to manually specify the AC point from the original
    image on the command line.,
    flag: --forceACPoint %s
forceHoughEyeDetectorReportFailure: (a boolean)
    , Flag indicates whether the Hough eye detector should report
    failure,
    flag: --forceHoughEyeDetectorReportFailure
forcePCPoint: (a list of items which are a float)
    , Use this flag to manually specify the PC point from the original
    image on the command line.,
    flag: --forcePCPoint %s
forceRPPPoint: (a list of items which are a float)
    , Use this flag to manually specify the RP point from the original
    image on the command line.,
    flag: --forceRPPPoint %s
forceVN4Point: (a list of items which are a float)
    , Use this flag to manually specify the VN4 point from the original
    image on the command line.,
    flag: --forceVN4Point %s
houghEyeDetectorMode: (an integer (int or long))
    , This flag controls the mode of Hough eye detector. By default,
    value of 1 is for T1W images, while the value of 0 is for T2W and PD
    images.,
    flag: --houghEyeDetectorMode %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputLandmarksEMSP: (an existing file name)
    , The filename for the new subject-specific landmark definition file
    in the same format produced by Slicer3 (in .fcsv) with the landmarks
    in the estimated MSP aligned space to be loaded. The detector will
    only process landmarks not enlisted on the file.,
    flag: --inputLandmarksEMSP %s
inputTemplateModel: (an existing file name)
    User-specified template model.,
    flag: --inputTemplateModel %s
inputVolume: (an existing file name)
    Input image in which to find ACPC points
    flag: --inputVolume %s

```

```

interpolationMode: ('NearestNeighbor' or 'Linear' or
    'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
    'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
    Type of interpolation to be used when applying transform to moving
    volume. Options are Linear, ResampleInPlace, NearestNeighbor,
    BSpline, or WindowedSinc
    flag: --interpolationMode %s
mspQualityLevel: (an integer (int or long))
    , Flag cotrols how aggressive the MSP is estimated. 0=quick estimate
    (9 seconds), 1=normal estimate (11 seconds), 2=great estimate (22
    seconds), 3=best estimate (58 seconds), NOTE: -1= Prealigned so no
    estimate!.,
    flag: --mspQualityLevel %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
otsuPercentileThreshold: (a float)
    , This is a parameter to FindLargestForegroundFilledMask, which is
    employed when acLowerBound is set and an
    outputUntransformedClippedVolume is requested.,
    flag: --otsuPercentileThreshold %f
outputLandmarksInACPCAlignedSpace: (a boolean or a file name)
    , The filename for the new subject-specific landmark definition file
    in the same format produced by Slicer3 (.fcsv) with the landmarks in
    the output image space (the detected RP, AC, PC, and VN4) in it to
    be written.,
    flag: --outputLandmarksInACPCAlignedSpace %s
outputLandmarksInInputSpace: (a boolean or a file name)
    , The filename for the new subject-specific landmark definition file
    in the same format produced by Slicer3 (.fcsv) with the landmarks in
    the original image space (the detected RP, AC, PC, and VN4) in it to
    be written.,
    flag: --outputLandmarksInInputSpace %s
outputMRML: (a boolean or a file name)
    , The filename for the new subject-specific scene definition file in
    the same format produced by Slicer3 (in .mrml format). Only the
    components that were specified by the user on command line would be
    generated. Compatible components include inputVolume, outputVolume,
    outputLandmarksInInputSpace, outputLandmarksInACPCAlignedSpace, and
    outputTransform.,
    flag: --outputMRML %s
outputResampledVolume: (a boolean or a file name)
    ACPC-aligned output image in a resampled unifor space. Currently
    this is a 1mm, 256^3, Identity direction image.
    flag: --outputResampledVolume %s
outputTransform: (a boolean or a file name)
    The filename for the original space to ACPC alignment to be written
    (in .h5 format).,
    flag: --outputTransform %s
outputUntransformedClippedVolume: (a boolean or a file name)
    Output image in which to store neck-clipped input image, with the
    use of --acLowerBound and maybe --cutOutHeadInUntransformedVolume.
    flag: --outputUntransformedClippedVolume %s
outputVerificationScript: (a boolean or a file name)
    , The filename for the Slicer3 script that verifies the aligned
    landmarks against the aligned image file. This will happen only in
    conjunction with saveOutputLandmarks and an outputVolume.,
    flag: --outputVerificationScript %s

```



```

outputVolume: (a boolean or a file name)
    ACPC-aligned output image with the same voxels, but updated origin,
    and direction cosign so that the AC point would fall at the physical
    location (0.0,0.0,0.0), and the mid-sagittal plane is the plane where
    physical L/R coordinate is 0.0.
    flag: --outputVolume %s
rVN4: (a float)
    , Search radius for VN4 in unit of mm,
    flag: --rVN4 %f
rac: (a float)
    , Search radius for AC in unit of mm,
    flag: --rac %f
rescaleIntensities: (a boolean)
    , Flag to turn on rescaling image intensities on input.,
    flag: --rescaleIntensities
rescaleIntensitiesOutputRange: (a list of items which are an integer
    (int or long))
    , This pair of integers gives the lower and upper bounds on the
    signal portion of the output image. Out-of-field voxels are taken
    from BackgroundFillValue.,
    flag: --rescaleIntensitiesOutputRange %s
resultsDir: (a boolean or a directory name)
    , The directory for the debugging images to be written.,
    flag: --resultsDir %s
rmpj: (a float)
    , Search radius for MPJ in unit of mm,
    flag: --rmpj %f
rpc: (a float)
    , Search radius for PC in unit of mm,
    flag: --rpc %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
trimRescaledIntensities: (a float)
    , Turn on clipping the rescaled image one-tailed on input. Units of
    standard deviations above the mean. Very large values are very
    permissive. Non-positive value turns clipping off. Defaults to
    removing 0.00001 of a normal tail above the mean.,
    flag: --trimRescaledIntensities %f
verbose: (a boolean)
    , Show more verbose output,
    flag: --verbose
writeBranded2DImage: (a boolean or a file name)
    , The filename for the 2D .png branded midline debugging image. This
    will happen only in conjunction with requesting an outputVolume.,
    flag: --writeBranded2DImage %s
writedebuggingImagesLevel: (an integer (int or long))
    , This flag controls if debugging images are produced. By default
    value of 0 is no images. Anything greater than zero will be
    increasing level of debugging images.,
    flag: --writedebuggingImagesLevel %d

```

Outputs:

```

outputLandmarksInACPCAlignedSpace: (an existing file name)
    , The filename for the new subject-specific landmark definition file
    in the same format produced by Slicer3 (.fcsv) with the landmarks in

```

```

    the output image space (the detected RP, AC, PC, and VN4) in it to
    be written.,
outputLandmarksInInputSpace: (an existing file name)
    , The filename for the new subject-specific landmark definition file
    in the same format produced by Slicer3 (.fcsv) with the landmarks in
    the original image space (the detected RP, AC, PC, and VN4) in it to
    be written.,
outputMRML: (an existing file name)
    , The filename for the new subject-specific scene definition file in
    the same format produced by Slicer3 (in .mrml format). Only the
    components that were specified by the user on command line would be
    generated. Compatible components include inputVolume, outputVolume,
    outputLandmarksInInputSpace, outputLandmarksInACPCAlignedSpace, and
    outputTransform.,
outputResampledVolume: (an existing file name)
    ACPC-aligned output image in a resampled unifor space. Currently
    this is a 1mm, 256^3, Identity direction image.
outputTransform: (an existing file name)
    The filename for the original space to ACPC alignment to be written
    (in .h5 format).,
outputUntransformedClippedVolume: (an existing file name)
    Output image in which to store neck-clipped input image, with the
    use of --acLowerBound and maybe --cutOutHeadInUntransformedVolume.
outputVerificationScript: (an existing file name)
    , The filename for the Slicer3 script that verifies the aligned
    landmarks against the aligned image file. This will happen only in
    conjunction with saveOutputLandmarks and an outputVolume.,
outputVolume: (an existing file name)
    ACPC-aligned output image with the same voxels, but updated origin,
    and direction cosign so that the AC point would fall at the physical
    location (0.0,0.0,0.0), and the mid-sagittal plane is the plane where
    physical L/R coordinate is 0.0.
resultsDir: (an existing directory name)
    , The directory for the debugging images to be written.,
writeBranded2DImage: (an existing file name)
    , The filename for the 2D .png branded midline debugging image. This
    will happen only in conjunction with requesting an outputVolume.,

```

BRAINSCreateLabelMapFromProbabilityMaps

[Link to code](#)

Wraps command **** BRAINSCreateLabelMapFromProbabilityMaps ****

title: Create Label Map From Probability Maps (BRAINS)

category: Segmentation.Specialized

description: Given A list of Probability Maps, generate a LabelMap.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
cleanLabelVolume: (a boolean or a file name)
    the foreground labels volume
    flag: --cleanLabelVolume %s
dirtyLabelVolume: (a boolean or a file name)

```

```

    the labels prior to cleaning
    flag: --dirtyLabelVolume %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
foregroundPriors: (a list of items which are an integer (int or
    long))
    A list: For each Prior Label, 1 if foreground, 0 if background
    flag: --foregroundPriors %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inclusionThreshold: (a float)
    tolerance for inclusion
    flag: --inclusionThreshold %f
inputProbabilityVolume: (a list of items which are an existing file
    name)
    The list of probobabilityimages.
    flag: --inputProbabilityVolume %s...
nonAirRegionMask: (an existing file name)
    a mask representing the 'NonAirRegion' -- Just force pixels in this
    region to zero
    flag: --nonAirRegionMask %s
priorLabelCodes: (a list of items which are an integer (int or long))
    A list of PriorLabelCode values used for coding the output label
    images
    flag: --priorLabelCodes %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

cleanLabelVolume: (an existing file name)
    the foreground labels volume
dirtyLabelVolume: (an existing file name)
    the labels prior to cleaning

```

BRAINSCut

[Link to code](#)

Wraps command **** BRAINSCut ****

title: BRAINSCut (BRAINS)

category: Segmentation.Specialized

description: Automatic Segmentation using neural networks

version: 1.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Vince Magnotta, Hans Johnson, Greg Harris, Kent Williams, Eunyoung Regina Kim

Inputs:

[Mandatory]

[Optional]

NoTrainingVectorShuffling: (a boolean)

If this flag is on, there will be no shuffling.

```
    flag: --NoTrainingVectorShuffling
applyModel: (a boolean)
    apply the neural net
    flag: --applyModel
args: (a unicode string)
    Additional parameters to the command
    flag: %s
computeSSEOn: (a boolean)
    compute Sum of Square Error (SSE) along the trained model until the
    number of iteration given in the modelConfigurationFilename file
    flag: --computeSSEOn
createVectors: (a boolean)
    create vectors for training neural net
    flag: --createVectors
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
generateProbability: (a boolean)
    Generate probability map
    flag: --generateProbability
histogramEqualization: (a boolean)
    A Histogram Equalization process could be added to the
    creating/applying process from Subject To Atlas. Default is false,
    which generate input vectors without Histogram Equalization.
    flag: --histogramEqualization
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
method: ('RandomForest' or 'ANN')
    flag: --method %s
modelConfigurationFilename: (an existing file name)
    XML File defining BRAINSCut parameters
    flag: --modelConfigurationFilename %s
modelFilename: (a unicode string)
    model file name given from user (not by xml configuration file)
    flag: --modelFilename %s
multiStructureThreshold: (a boolean)
    multiStructureThreshold module to deal with overlapping area
    flag: --multiStructureThreshold
netConfiguration: (an existing file name)
    XML File defining BRAINSCut parameters. OLD NAME. PLEASE USE
    modelConfigurationFilename instead.
    flag: --netConfiguration %s
numberOfTrees: (an integer (int or long))
    Random tree: number of trees. This is to be used when only one
    model with specified depth wish to be created.
    flag: --numberOfTrees %d
randomTreeDepth: (an integer (int or long))
    Random tree depth. This is to be used when only one model with
    specified depth wish to be created.
    flag: --randomTreeDepth %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
trainModel: (a boolean)
    train the neural net
```

```

        flag: --trainModel
trainModelIndex: (an integer (int or long))
    Starting iteration for training
        flag: --trainModelIndex %d
validate: (a boolean)
    validate data set. Just need for the first time run ( This is for
    validation of xml file and not working yet )
        flag: --validate
verbose: (an integer (int or long))
    print out some debugging information
        flag: --verbose %d

```

Outputs:

None

BRAINSMultiSTAPLE

[Link to code](#)

Wraps command **** BRAINSMultiSTAPLE ****

title: Create best representative label map)

category: Segmentation.Specialized

description: given a list of label map images, create a representative/average label map.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputCompositeT1Volume: (an existing file name)
    Composite T1, all label maps transofrmed into the space for this
    image.
    flag: --inputCompositeT1Volume %s
inputLabelVolume: (a list of items which are an existing file name)
    The list of proobabilityimages.
    flag: --inputLabelVolume %s...
inputTransform: (a list of items which are an existing file name)
    transforms to apply to label volumes
    flag: --inputTransform %s...
labelForUndecidedPixels: (an integer (int or long))
    Label for undecided pixels
    flag: --labelForUndecidedPixels %d
outputConfusionMatrix: (a boolean or a file name)
    Confusion Matrix
    flag: --outputConfusionMatrix %s
outputMultiSTAPLE: (a boolean or a file name)
    the MultiSTAPLE average of input label volumes
    flag: --outputMultiSTAPLE %s
resampledVolumePrefix: (a unicode string)

```

```

        if given, write out resampled volumes with this prefix
        flag: --resampledVolumePrefix %s
skipResampling: (a boolean)
    Omit resampling images into reference space
    flag: --skipResampling
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputConfusionMatrix: (an existing file name)
    Confusion Matrix
outputMultiSTAPLE: (an existing file name)
    the MultiSTAPLE average of input label volumes

```

BRAINSROIAuto[Link to code](#)

Wraps command **** BRAINSROIAuto ****

title: Foreground masking (BRAINS)

category: Segmentation.Specialized

description: This program is used to create a mask over the most prominent foreground region in an image. This is accomplished via a combination of otsu thresholding and a closing operation. More documentation is available here: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ForegroundMasking>.

version: 2.4.1

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Hans J. Johnson, hans-johnson -at- uiowa.edu, <http://www.psychiatry.uiowa.edu>

acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); Gregory Harris(1), Vincent Magnotta(1,2,3); Andriy Fedorov(5), fedorov -at- bwh.harvard.edu (Slicer integration); (1=University of Iowa Department of Psychiatry, 2=University of Iowa Department of Radiology, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering, 5=Surgical Planning Lab, Harvard)

Inputs:

```

[Mandatory]

[Optional]
ROIAutoDilateSize: (a float)
    This flag is only relevant when using ROIAUTO mode for initializing
    masks. It defines the final dilation size to capture a bit of
    background outside the tissue region. At setting of 10mm has been
    shown to help regularize a BSpline registration type so that there
    is some background constraints to match the edges of the head
    better.
    flag: --ROIAutoDilateSize %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
closingSize: (a float)
    The Closing Size (in millimeters) for largest connected filled mask.
    This value is divided by image spacing and rounded to the next
    largest voxel number.
    flag: --closingSize %f
cropOutput: (a boolean)
    The inputVolume cropped to the region of the ROI mask.

```

```

    flag: --cropOutput
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    The input image for finding the largest region filled mask.
    flag: --inputVolume %s
maskOutput: (a boolean)
    The inputVolume multiplied by the ROI mask.
    flag: --maskOutput
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
otsuPercentileThreshold: (a float)
    Parameter to the Otsu threshold algorithm.
    flag: --otsuPercentileThreshold %f
outputROIMaskVolume: (a boolean or a file name)
    The ROI automatically found from the input image.
    flag: --outputROIMaskVolume %s
outputVolume: (a boolean or a file name)
    The inputVolume with optional [maskOutput|cropOutput] to the region
    of the brain mask.
    flag: --outputVolume %s
outputVolumePixelType: ('float' or 'short' or 'ushort' or 'int' or
    'uint' or 'uchar')
    The output image Pixel Type is the scalar datatype for
    representation of the Output Volume.
    flag: --outputVolumePixelType %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thresholdCorrectionFactor: (a float)
    A factor to scale the Otsu algorithm's result threshold, in case
    clipping mangles the image.
    flag: --thresholdCorrectionFactor %f

```

Outputs:

```

outputROIMaskVolume: (an existing file name)
    The ROI automatically found from the input image.
outputVolume: (an existing file name)
    The inputVolume with optional [maskOutput|cropOutput] to the region
    of the brain mask.

```

BinaryMaskEditorBasedOnLandmarks[Link to code](#)Wraps command **** BinaryMaskEditorBasedOnLandmarks ****

title: BRAINS Binary Mask Editor Based On Landmarks(BRAINS)

category: Segmentation.Specialized

version: 1.0

documentation-url: <http://www.nitrc.org/projects/brainscdetector/>

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputBinaryVolume: (an existing file name)
    Input binary image in which to be edited
    flag: --inputBinaryVolume %s
inputLandmarkNames: (a list of items which are a unicode string)
    A target input landmark name to be edited. This should be listed in
    the inputLandmakrFilename Given.
    flag: --inputLandmarkNames %s
inputLandmarkNamesForObliquePlane: (a list of items which are a
    unicode string)
    Three subset landmark names of inputLandmarksFilename for a oblique
    plane computation. The plane computed for binary volume editing.
    flag: --inputLandmarkNamesForObliquePlane %s
inputLandmarksFilename: (an existing file name)
    The filename for the landmark definition file in the same format
    produced by Slicer3 (.fcsv).
    flag: --inputLandmarksFilename %s
outputBinaryVolume: (a boolean or a file name)
    Output binary image in which to be edited
    flag: --outputBinaryVolume %s
setCutDirectionForLandmark: (a list of items which are a unicode
    string)
    Setting the cutting out direction of the input binary image to the
    one of anterior, posterior, left, right, superior or posterior.
    (ENUMERATION: ANTERIOR, POSTERIOR, LEFT, RIGHT, SUPERIOR, POSTERIOR)
    flag: --setCutDirectionForLandmark %s
setCutDirectionForObliquePlane: (a list of items which are a unicode
    string)
    If this is true, the mask will be thresholded out to the direction
    of inferior, posterior, and/or left. Default behavior is that
    cutting out to the direction of superior, anterior and/or right.
    flag: --setCutDirectionForObliquePlane %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputBinaryVolume: (an existing file name)
    Output binary image in which to be edited

```

ESLR[Link to code](#)Wraps command **** ESLR ****

title: Clean Contiguous Label Map (BRAINS)

category: Segmentation.Specialized

description: From a range of label map values, extract the largest contiguous region of those labels

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
closingSize: (an integer (int or long))
    The closing size for hole filling.
    flag: --closingSize %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
high: (an integer (int or long))
    The higher bound of the labels to be used.
    flag: --high %d
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input Label Volume
    flag: --inputVolume %s
low: (an integer (int or long))
    The lower bound of the labels to be used.
    flag: --low %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
openingSize: (an integer (int or long))
    The opening size for hole filling.
    flag: --openingSize %d
outputVolume: (a boolean or a file name)
    Output Label Volume
    flag: --outputVolume %s
preserveOutside: (a boolean)
    For values outside the specified range, preserve those values.
    flag: --preserveOutside
safetySize: (an integer (int or long))
    The safetySize size for the clipping region.
    flag: --safetySize %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Output Label Volume
```

interfaces.semtools.utilities.brains

BRAINSAlignMSP

[Link to code](#)

Wraps command **** BRAINSAlignMSP ****

title: Align Mid Saggital Brain (BRAINS)

category: Utilities.BRAINS

description: Resample an image into ACPC alignment ACPCDetect

Inputs:

```

[Mandatory]

[Optional]
BackgroundFillValue: (a unicode string)
    Fill the background of image with specified short int value. Enter
    number or use BIGNEG for a large negative number.
    flag: --BackgroundFillValue %s
OutputresampleMSP: (a boolean or a file name)
    , The image to be output.,
    flag: --OutputresampleMSP %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    , The Image to be resampled,
    flag: --inputVolume %s
interpolationMode: ('NearestNeighbor' or 'Linear' or
    'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
    'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
    Type of interpolation to be used when applying transform to moving
    volume. Options are Linear, ResampleInPlace, NearestNeighbor,
    BSpline, or WindowedSinc
    flag: --interpolationMode %s
mspQualityLevel: (an integer (int or long))
    , Flag cotrols how aggressive the MSP is estimated. 0=quick estimate
    (9 seconds), 1=normal estimate (11 seconds), 2=great estimate (22
    seconds), 3=best estimate (58 seconds)..
    flag: --mspQualityLevel %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
rescaleIntensities: (a boolean)
    , Flag to turn on rescaling image intensities on input.,
    flag: --rescaleIntensities
rescaleIntensitiesOutputRange: (a list of items which are an integer
    (int or long))
    , This pair of integers gives the lower and upper bounds on the
    signal portion of the output image. Out-of-field voxels are taken
    from BackgroundFillValue.,
    flag: --rescaleIntensitiesOutputRange %s

```

```

resultsDir: (a boolean or a directory name)
    , The directory for the results to be written.,
    flag: --resultsDir %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
trimRescaledIntensities: (a float)
    , Turn on clipping the rescaled image one-tailed on input. Units of
    standard deviations above the mean. Very large values are very
    permissive. Non-positive value turns clipping off. Defaults to
    removing 0.00001 of a normal tail above the mean.,
    flag: --trimRescaledIntensities %f
verbose: (a boolean)
    , Show more verbose output,
    flag: --verbose
writedebuggingImagesLevel: (an integer (int or long))
    , This flag controls if debugging images are produced. By default
    value of 0 is no images. Anything greater than zero will be
    increasing level of debugging images.,
    flag: --writedebuggingImagesLevel %d

```

Outputs:

```

OutputresampleMSP: (an existing file name)
    , The image to be output.,
resultsDir: (an existing directory name)
    , The directory for the results to be written.,

```

BRAINSClipInferior[Link to code](#)Wraps command **** BRAINSClipInferior ****

title: Clip Inferior of Center of Brain (BRAINS)

category: Utilities.BRAINS

description: This program will read the inputVolume as a short int image, write the BackgroundFillValue everywhere inferior to the lower bound, and write the resulting clipped short int image in the outputVolume.

version: 1.0

Inputs:

```

[Mandatory]

[Optional]
BackgroundFillValue: (a unicode string)
    Fill the background of image with specified short int value. Enter
    number or use BIGNEG for a large negative number.
    flag: --BackgroundFillValue %s
acLowerBound: (a float)
    , When the input image to the output image, replace the image with
    the BackgroundFillValue everywhere below the plane This Far in
    physical units (millimeters) below (inferior to) the AC point
    (assumed to be the voxel field middle.) The oversize default was
    chosen to have no effect. Based on visualizing a thousand masks in
    the IPIG study, we recommend a limit no smaller than 80.0 mm.,
    flag: --acLowerBound %f
args: (a unicode string)
    Additional parameters to the command

```

```

    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input image to make a clipped short int copy from.
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    Output image, a short int copy of the upper portion of the input
    image, filled with BackgroundFillValue.
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Output image, a short int copy of the upper portion of the input
    image, filled with BackgroundFillValue.

```

BRAINSConstellationModeler[Link to code](#)Wraps command **** BRAINSConstellationModeler ******title:** Generate Landmarks Model (BRAINS)**category:** Utilities.BRAINS**description:** Train up a model for BRAINSConstellationDetector**Inputs:**

```

[Mandatory]

[Optional]
BackgroundFillValue: (a unicode string)
    Fill the background of image with specified short int value. Enter
    number or use BIGNEG for a large negative number.
    flag: --BackgroundFillValue %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputTrainingList: (an existing file name)
    , Setup file, giving all parameters for training up a template model
    for each landmark.,

```

```

    flag: --inputTrainingList %s
mspQualityLevel: (an integer (int or long))
    , Flag cotrols how aggressive the MSP is estimated. 0=quick estimate
    (9 seconds), 1=normal estimate (11 seconds), 2=great estimate (22
    seconds), 3=best estimate (58 seconds).,
    flag: --mspQualityLevel %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
optimizedLandmarksFilenameExtender: (a unicode string)
    , If the trainingList is (indexFullPathName) and contains landmark
    data filenames [path]/[filename].fcsv , make the optimized landmarks
    filenames out of [path]/[filename](thisExtender) and the optimized
    version of the input trainingList out of
    (indexFullPathName)(thisExtender) , when you rewrite all the
    landmarks according to the saveOptimizedLandmarks flag.,
    flag: --optimizedLandmarksFilenameExtender %s
outputModel: (a boolean or a file name)
    , The full filename of the output model file.,
    flag: --outputModel %s
rescaleIntensities: (a boolean)
    , Flag to turn on rescaling image intensities on input.,
    flag: --rescaleIntensities
rescaleIntensitiesOutputRange: (a list of items which are an integer
    (int or long))
    , This pair of integers gives the lower and upper bounds on the
    signal portion of the output image. Out-of-field voxels are taken
    from BackgroundFillValue.,
    flag: --rescaleIntensitiesOutputRange %s
resultsDir: (a boolean or a directory name)
    , The directory for the results to be written.,
    flag: --resultsDir %s
saveOptimizedLandmarks: (a boolean)
    , Flag to make a new subject-specific landmark definition file in
    the same format produced by Slicer3 with the optimized landmark (the
    detected RP, AC, and PC) in it. Useful to tighten the variances in
    the ConstellationModeler.,
    flag: --saveOptimizedLandmarks
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
trimRescaledIntensities: (a float)
    , Turn on clipping the rescaled image one-tailed on input. Units of
    standard deviations above the mean. Very large values are very
    permissive. Non-positive value turns clipping off. Defaults to
    removing 0.00001 of a normal tail above the mean.,
    flag: --trimRescaledIntensities %f
verbose: (a boolean)
    , Show more verbose output,
    flag: --verbose
writedebuggingImagesLevel: (an integer (int or long))
    , This flag controls if debugging images are produced. By default
    value of 0 is no images. Anything greater than zero will be
    increasing level of debugging images.,
    flag: --writedebuggingImagesLevel %d

```

Outputs:

```
outputModel: (an existing file name)
    , The full filename of the output model file.,
resultsDir: (an existing directory name)
    , The directory for the results to be written.,
```

BRAINSEyeDetector

[Link to code](#)

Wraps command **** BRAINSEyeDetector ****

title: Eye Detector (BRAINS)

category: Utilities.BRAINS

version: 1.0

documentation-url: <http://www.nitrc.org/projects/brainscdetector/>

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debugDir: (a unicode string)
    A place for debug information
    flag: --debugDir %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    The input volume
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
    The output volume
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    The output volume
```

BRAINSInitializedControlPoints

[Link to code](#)

Wraps command **** BRAINSInitializedControlPoints ****

title: Initialized Control Points (BRAINS)

category: Utilities.BRAINS

description: Outputs bspline control points as landmarks

version: 0.1.0.\$Revision: 916 \$(alpha)

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Mark Scully

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Additional support for Mark Scully and Hans Johnson at the University of Iowa.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input Volume
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputLandmarksFile: (a unicode string)
    Output filename
    flag: --outputLandmarksFile %s
outputVolume: (a boolean or a file name)
    Output Volume
    flag: --outputVolume %s
permuteOrder: (a list of items which are an integer (int or long))
    The permutation order for the images. The default is 0,1,2 (i.e. no
    permutation)
    flag: --permuteOrder %s
splineGridSize: (a list of items which are an integer (int or long))
    The number of subdivisions of the BSpline Grid to be centered on the
    image space. Each dimension must have at least 3 subdivisions for
    the BSpline to be correctly computed.
    flag: --splineGridSize %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Output Volume
```

BRAINSLandmarkInitializer

[Link to code](#)

Wraps command `** BRAINSLandmarkInitializer **`

title: BRAINSLandmarkInitializer

category: Utilities.BRAINS

description: Create transformation file (*mat) from a pair of landmarks (*fcsv) files.

version: 1.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Eunyoung Regina Kim

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputFixedLandmarkFilename: (an existing file name)
    input fixed landmark. *.fcsv
    flag: --inputFixedLandmarkFilename %s
inputMovingLandmarkFilename: (an existing file name)
    input moving landmark. *.fcsv
    flag: --inputMovingLandmarkFilename %s
inputWeightFilename: (an existing file name)
    Input weight file name for landmarks. Higher weighted landmark will
    be considered more heavily. Weights are propotional, that is the
    magnitude of weights will be normalized by its minimum and maximum
    value.
    flag: --inputWeightFilename %s
outputTransformFilename: (a boolean or a file name)
    output transform file name (ex: ./outputTransform.mat)
    flag: --outputTransformFilename %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputTransformFilename: (an existing file name)
    output transform file name (ex: ./outputTransform.mat)
```

BRAINSLinearModelerEPCA

[Link to code](#)

Wraps command **** BRAINSLinearModelerEPCA ****

title: Landmark Linear Modeler (BRAINS)

category: Utilities.BRAINS

description: Training linear model using EPCA. Implementation based on my MS thesis, "A METHOD FOR AUTOMATED LANDMARK CONSTELLATION DETECTION USING EVOLUTIONARY PRINCIPAL COMPONENTS AND STATISTICAL SHAPE MODELS"

version: 1.0

documentation-url: <http://www.nitrc.org/projects/brainscdetector/>

Inputs:


```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputTrainingList: (an existing file name)
    Input Training Landmark List Filename,
    flag: --inputTrainingList %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

None

BRAINSLmkTransform

[Link to code](#)

Wraps command **** BRAINSLmkTransform ****

title: Landmark Transform (BRAINS)

category: Utilities.BRAINS

description: This utility program estimates the affine transform to align the fixed landmarks to the moving landmarks, and then generate the resampled moving image to the same physical space as that of the reference image.

version: 1.0

documentation-url: <http://www.nitrc.org/projects/brainscdetector/>

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputFixedLandmarks: (an existing file name)
    Input Fixed Landmark list file in fcsv,
    flag: --inputFixedLandmarks %s

```

```

inputMovingLandmarks: (an existing file name)
    Input Moving Landmark list file in fcsv,
    flag: --inputMovingLandmarks %s
inputMovingVolume: (an existing file name)
    The filename of input moving volume
    flag: --inputMovingVolume %s
inputReferenceVolume: (an existing file name)
    The filename of the reference volume
    flag: --inputReferenceVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputAffineTransform: (a boolean or a file name)
    The filename for the estimated affine transform,
    flag: --outputAffineTransform %s
outputResampledVolume: (a boolean or a file name)
    The filename of the output resampled volume
    flag: --outputResampledVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputAffineTransform: (an existing file name)
    The filename for the estimated affine transform,
outputResampledVolume: (an existing file name)
    The filename of the output resampled volume

```

BRAINSMush[Link to code](#)

Wraps command **** BRAINSMush ****

title: Brain Extraction from T1/T2 image (BRAINS)

category: Utilities.BRAINS

description: This program: 1) generates a weighted mixture image optimizing the mean and variance and 2) produces a mask of the brain volume

version: 0.1.0.\$Revision: 1.4 \$(alpha)

documentation-url: <http://mri.radiology.uiowa.edu>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool is a modification by Steven Dunn of a program developed by Greg Harris and Ron Pierson.

acknowledgements: This work was developed by the University of Iowa Departments of Radiology and Psychiatry. This software was supported in part of NIH/NINDS award NS050568.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
boundingBoxSize: (a list of items which are an integer (int or long))
    Size of the cubic bounding box mask used when no brain mask is
    present
    flag: --boundingBoxSize %s
boundingBoxStart: (a list of items which are an integer (int or

```

```

        long))
        XYZ point-coordinate for the start of the cubic bounding box mask
        used when no brain mask is present
        flag: --boundingBoxStart %s
desiredMean: (a float)
        Desired mean within the mask for weighted sum of both images.
        flag: --desiredMean %f
desiredVariance: (a float)
        Desired variance within the mask for weighted sum of both images.
        flag: --desiredVariance %f
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
inputFirstVolume: (an existing file name)
        Input image (1) for mixture optimization
        flag: --inputFirstVolume %s
inputMaskVolume: (an existing file name)
        Input label image for mixture optimization
        flag: --inputMaskVolume %s
inputSecondVolume: (an existing file name)
        Input image (2) for mixture optimization
        flag: --inputSecondVolume %s
lowerThresholdFactor: (a float)
        Lower threshold factor for defining the brain mask
        flag: --lowerThresholdFactor %f
lowerThresholdFactorPre: (a float)
        Lower threshold factor for finding an initial brain mask
        flag: --lowerThresholdFactorPre %f
numberOfThreads: (an integer (int or long))
        Explicitly specify the maximum number of threads to use.
        flag: --numberOfThreads %d
outputMask: (a boolean or a file name)
        The brain volume mask generated from the MUSH image
        flag: --outputMask %s
outputVolume: (a boolean or a file name)
        The MUSH image produced from the T1 and T2 weighted images
        flag: --outputVolume %s
outputWeightsFile: (a boolean or a file name)
        Output Weights File
        flag: --outputWeightsFile %s
seed: (a list of items which are an integer (int or long))
        Seed Point for Brain Region Filling
        flag: --seed %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored
upperThresholdFactor: (a float)
        Upper threshold factor for defining the brain mask
        flag: --upperThresholdFactor %f
upperThresholdFactorPre: (a float)
        Upper threshold factor for finding an initial brain mask
        flag: --upperThresholdFactorPre %f

```

Outputs:

```
outputMask: (an existing file name)
    The brain volume mask generated from the MUSH image
outputVolume: (an existing file name)
    The MUSH image produced from the T1 and T2 weighted images
outputWeightsFile: (an existing file name)
    Output Weights File
```

BRAINSSnapShotWriter

[Link to code](#)

Wraps command **BRAINSSnapShotWriter**

title: BRAINSSnapShotWriter

category: Utilities.BRAINS

description: Create 2D snapshot of input images. Mask images are color-coded

version: 1.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Eunyoung Regina Kim

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputBinaryVolumes: (a list of items which are an existing file name)
    Input mask (binary) volume list to be extracted as 2D image.
    Multiple input is possible.
    flag: --inputBinaryVolumes %s...
inputPlaneDirection: (a list of items which are an integer (int or
    long))
    Plane to display. In general, 0=sagittal, 1=coronal, and 2=axial
    plane.
    flag: --inputPlaneDirection %s
inputSliceToExtractInIndex: (a list of items which are an integer
    (int or long))
    2D slice number of input images. For size of 256*256*256 image, 128
    is usually used.
    flag: --inputSliceToExtractInIndex %s
inputSliceToExtractInPercent: (a list of items which are an integer
    (int or long))
    2D slice number of input images. Percentage input from 0%-100%. (ex.
    --inputSliceToExtractInPercent 50,50,50
    flag: --inputSliceToExtractInPercent %s
inputSliceToExtractInPhysicalPoint: (a list of items which are a
    float)
    2D slice number of input images. For autoWorkUp output, which AC-PC
    aligned, 0,0,0 will be the center.
    flag: --inputSliceToExtractInPhysicalPoint %s
inputVolumes: (a list of items which are an existing file name)
```

```

    Input image volume list to be extracted as 2D image. Multiple input
    is possible. At least one input is required.
    flag: --inputVolumes %s...
outputFilename: (a boolean or a file name)
    2D file name of input images. Required.
    flag: --outputFilename %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputFilename: (an existing file name)
    2D file name of input images. Required.

```

BRAINSTransformConvert[Link to code](#)Wraps command **** BRAINSTransformConvert ****

title: BRAINS Transform Convert

category: Utilities.BRAINS

description: Convert ITK transforms to higher order transforms

version: 1.0

documentation-url: A utility to convert between transform file formats.

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Hans J. Johnson,Kent Williams, Ali Ghayoor

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
displacementVolume: (a boolean or a file name)
    flag: --displacementVolume %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputTransform: (an existing file name)
    flag: --inputTransform %s
outputPrecisionType: ('double' or 'float')
    Precision type of the output transform. It can be either single
    precision or double precision
    flag: --outputPrecisionType %s
outputTransform: (a boolean or a file name)
    flag: --outputTransform %s
outputTransformType: ('Affine' or 'VersorRigid' or 'ScaleVersor' or
    'ScaleSkewVersor' or 'DisplacementField' or 'Same')
    The target transformation type. Must be conversion-compatible with
    the input transform type
    flag: --outputTransformType %s

```

```
referenceVolume: (an existing file name)
    flag: --referenceVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
displacementVolume: (an existing file name)
outputTransform: (an existing file name)
```

BRAINSTrimForegroundInDirection[Link to code](#)**Wraps command** **** BRAINSTrimForegroundInDirection ******title:** Trim Foreground In Direction (BRAINS)**category:** Utilities.BRAINS**description:** This program will trim off the neck and also air-filling noise from the inputImage.**version:** 0.1**documentation-url:** <http://www.nitrc.org/projects/art/>**Inputs:**

```
[Mandatory]

[Optional]
BackgroundFillValue: (a unicode string)
    Fill the background of image with specified short int value. Enter
    number or use BIGNEG for a large negative number.
    flag: --BackgroundFillValue %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
closingSize: (an integer (int or long))
    , This is a parameter to FindLargestForegroundFilledMask,
    flag: --closingSize %d
directionCode: (an integer (int or long))
    , This flag chooses which dimension to compare. The sign lets you
    flip direction.,
    flag: --directionCode %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
headSizeLimit: (a float)
    , Use this to vary from the command line our search for how much
    upper tissue is head for the center-of-mass calculation. Units are
    CCs, not cubic millimeters.,
    flag: --headSizeLimit %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input image to trim off the neck (and also air-filling noise.)
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
```

```

    flag: --numberOfThreads %d
otsuPercentileThreshold: (a float)
    , This is a parameter to FindLargestForegroundFilledMask, which is
    employed to trim off air-filling noise.,
    flag: --otsuPercentileThreshold %f
outputVolume: (a boolean or a file name)
    Output image with neck and air-filling noise trimmed isotropic image
    with AC at center of image.
    flag: --outputVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Output image with neck and air-filling noise trimmed isotropic image
    with AC at center of image.

```

CleanUpOverlapLabels[Link to code](#)Wraps command **** CleanUpOverlapLabels ****

title: Clean Up Overla Labels

category: Utilities.BRAINS

description: Take a series of input binary images and clean up for those overlapped area. Binary volumes given first always wins out

version: 0.1.0

contributor: Eun Young Kim

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputBinaryVolumes: (a list of items which are an existing file name)
    The list of binary images to be checked and cleaned up. Order is
    important. Binary volume given first always wins out.
    flag: --inputBinaryVolumes %s...
outputBinaryVolumes: (a boolean or a list of items which are a file
    name)
    The output label map images, with integer values in it. Each label
    value specified in the inputLabels is combined into this output
    label map volume
    flag: --outputBinaryVolumes %s...
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately

```

```
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputBinaryVolumes: (a list of items which are an existing file
name)
The output label map images, with integer values in it. Each label
value specified in the inputLabels is combined into this output
label map volume
```

FindCenterOfBrain

[Link to code](#)

Wraps command **** FindCenterOfBrain ****

title: Center Of Brain (BRAINS)

category: Utilities.BRAINS

description: Finds the center point of a brain

version: 3.0.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Hans J. Johnson, hans-johnson -at- uiowa.edu, <http://www.psychiatry.uiowa.edu>

acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); (1=University of Iowa Department of Psychiatry, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
axis: (an integer (int or long))
    flag: --axis %d
backgroundValue: (an integer (int or long))
    flag: --backgroundValue %d
clippedImageMask: (a boolean or a file name)
    flag: --clippedImageMask %s
closingSize: (an integer (int or long))
    flag: --closingSize %d
debugAfterGridComputationsForegroundImage: (a boolean or a file name)
    flag: --debugAfterGridComputationsForegroundImage %s
debugClippedImageMask: (a boolean or a file name)
    flag: --debugClippedImageMask %s
debugDistanceImage: (a boolean or a file name)
    flag: --debugDistanceImage %s
debugGridImage: (a boolean or a file name)
    flag: --debugGridImage %s
debugTrimmedImage: (a boolean or a file name)
    flag: --debugTrimmedImage %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
generateDebugImages: (a boolean)
    flag: --generateDebugImages
headSizeEstimate: (a float)
```



```

        flag: --headSizeEstimate %f
headSizeLimit: (a float)
        flag: --headSizeLimit %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
imageMask: (an existing file name)
        flag: --imageMask %s
inputVolume: (an existing file name)
    The image in which to find the center.
        flag: --inputVolume %s
maximize: (a boolean)
        flag: --maximize
otsuPercentileThreshold: (a float)
        flag: --otsuPercentileThreshold %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

clippedImageMask: (an existing file name)
debugAfterGridComputationsForegroundImage: (an existing file name)
debugClippedImageMask: (an existing file name)
debugDistanceImage: (an existing file name)
debugGridImage: (an existing file name)
debugTrimmedImage: (an existing file name)

```

GenerateLabelMapFromProbabilityMap[Link to code](#)Wraps command **** GenerateLabelMapFromProbabilityMap ****

title: Label Map from Probability Images

category: Utilities.BRAINS

description: Given a list of probability maps for labels, create a discrete label map where only the highest probability region is used for the labeling.

version: 0.1

contributor: University of Iowa Department of Psychiatry, <http://www.psychiatry.uiowa.edu>

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolumes: (a list of items which are an existing file name)
    The Input probaiblity images to be computed for lable maps
    flag: --inputVolumes %s...

```

```
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputLabelVolume: (a boolean or a file name)
    The Input binary image for region of interest
    flag: --outputLabelVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputLabelVolume: (an existing file name)
    The Input binary image for region of interest
```

ImageRegionPlotter

[Link to code](#)

Wraps command **** ImageRegionPlotter ****

title: Write Out Image Intensities

category: Utilities.BRAINS

description: For Analysis

version: 0.1

contributor: University of Iowa Department of Psychiatry, <http://www.psychiatry.uiowa.edu>

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputBinaryROIVolume: (an existing file name)
    The Input binary image for region of interest
    flag: --inputBinaryROIVolume %s
inputLabelVolume: (an existing file name)
    The Label Image
    flag: --inputLabelVolume %s
inputVolume1: (an existing file name)
    The Input image to be computed for statistics
    flag: --inputVolume1 %s
inputVolume2: (an existing file name)
    The Input image to be computed for statistics
    flag: --inputVolume2 %s
numberOfHistogramBins: (an integer (int or long))
    the number of histogram levels
    flag: --numberOfHistogramBins %d
outputJointHistogramData: (a unicode string)
    output data file name
```

```

    flag: --outputJointHistogramData %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
useIntensityForHistogram: (a boolean)
    Create Intensity Joint Histogram instead of Quantile Joint
    Histogram
    flag: --useIntensityForHistogram
useROIAUTO: (a boolean)
    Use ROIAUTO to compute region of interest. This cannot be used with
    inputLabelVolume
    flag: --useROIAUTO
verbose: (a boolean)
    print debugging information,
    flag: --verbose

```

Outputs:

None

JointHistogram

[Link to code](#)

Wraps command **** JointHistogram ****

title: Write Out Image Intensities

category: Utilities.BRAINS

description: For Analysis

version: 0.1

contributor: University of Iowa Department of Psychiatry, <http://www.psychiatry.uiowa.edu>

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputMaskVolumeInXAxis: (an existing file name)
    Input mask volume for inputVolumeInXAxis. Histogram will be computed
    just for the masked region
    flag: --inputMaskVolumeInXAxis %s
inputMaskVolumeInYAxis: (an existing file name)
    Input mask volume for inputVolumeInYAxis. Histogram will be computed
    just for the masked region
    flag: --inputMaskVolumeInYAxis %s
inputVolumeInXAxis: (an existing file name)
    The Input image to be computed for statistics
    flag: --inputVolumeInXAxis %s
inputVolumeInYAxis: (an existing file name)

```

```

    The Input image to be computed for statistics
    flag: --inputVolumeInYAxis %s
outputJointHistogramImage: (a unicode string)
    output joint histogram image file name. Histogram is usually 2D
    image.
    flag: --outputJointHistogramImage %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    print debugging information,
    flag: --verbose

```

Outputs:

None

ShuffleVectorsModule

[Link to code](#)

Wraps command **** ShuffleVectorsModule ****

title: ShuffleVectors

category: Utilities.BRAINS

description: Automatic Segmentation using neural networks

version: 1.0

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Hans Johnson

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVectorFileBaseName: (an existing file name)
    input vector file name prefix. Usually end with .txt and header file
    has prost fix of .txt.hdr
    flag: --inputVectorFileBaseName %s
outputVectorFileBaseName: (a boolean or a file name)
    output vector file name prefix. Usually end with .txt and header
    file has prost fix of .txt.hdr
    flag: --outputVectorFileBaseName %s
resampleProportion: (a float)
    downsample size of 1 will be the same size as the input images,
    downsample size of 3 will throw 2/3 the vectors away.
    flag: --resampleProportion %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately

```

```
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVectorFileBaseName: (an existing file name)
    output vector file name prefix. Usually end with .txt and header
    file has prost fix of .txt.hdr
```

fcsv_to_hdf5[Link to code](#)Wraps command **** fcsv_to_hdf5 ****

title: fcsv_to_hdf5 (BRAINS)

category: Utilities.BRAINS

description: Convert a collection of fcsv files to a HDF5 format file

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
landmarkGlobPattern: (a unicode string)
    Glob pattern to select fcsv files
    flag: --landmarkGlobPattern %s
landmarkTypesList: (an existing file name)
    , file containing list of landmark types,
    flag: --landmarkTypesList %s
landmarksInformationFile: (a boolean or a file name)
    , name of HDF5 file to write matrices into,
    flag: --landmarksInformationFile %s
modelFile: (a boolean or a file name)
    , name of HDF5 file containing BRAINSConstellationDetector Model
    file (LLSMatrices, LLSMeans and LLSSearchRadii),
    flag: --modelFile %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
versionID: (a unicode string)
    , Current version ID. It should be match with the version of BCD
    that will be using the output model file,
    flag: --versionID %s
```

Outputs:

```
landmarksInformationFile: (an existing file name)
    , name of HDF5 file to write matrices into,
modelFile: (an existing file name)
    , name of HDF5 file containing BRAINSConstellationDetector Model
    file (LLSMatrices, LLSMeans and LLSSearchRadii),
```

insertMidACPCpoint

[Link to code](#)

Wraps command **** insertMidACPCpoint ****

title: MidACPC Landmark Insertion

category: Utilities.BRAINS

description: This program gets a landmark fcsv file and adds a new landmark as the midpoint between AC and PC points to the output landmark fcsv file

contributor: Ali Ghayoor

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputLandmarkFile: (an existing file name)
    Input landmark file (.fcsv)
    flag: --inputLandmarkFile %s
outputLandmarkFile: (a boolean or a file name)
    Output landmark file (.fcsv)
    flag: --outputLandmarkFile %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputLandmarkFile: (an existing file name)
    Output landmark file (.fcsv)
```

landmarksConstellationAligner

[Link to code](#)

Wraps command **** landmarksConstellationAligner ****

title: MidACPC Landmark Insertion

category: Utilities.BRAINS

description: This program converts the original landmark files to the acpc-aligned landmark files

contributor: Ali Ghayoor

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputLandmarksPaired: (an existing file name)
    Input landmark file (.fcsv)
    flag: --inputLandmarksPaired %s
outputLandmarksPaired: (a boolean or a file name)
    Output landmark file (.fcsv)
    flag: --outputLandmarksPaired %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputLandmarksPaired: (an existing file name)
    Output landmark file (.fcsv)
```

landmarksConstellationWeights[Link to code](#)**Wraps command** `** landmarksConstellationWeights **`**title:** Generate Landmarks Weights (BRAINS)**category:** Utilities.BRAINS**description:** Train up a list of Weights for the Landmarks in BRAINSConstellationDetector**Inputs:**

```
[Mandatory]

[Optional]
LLSModel: (an existing file name)
    Linear least squares model filename in HD5 format
    flag: --LLSModel %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputTemplateModel: (an existing file name)
    User-specified template model.,
    flag: --inputTemplateModel %s
inputTrainingList: (an existing file name)
```

```
, Setup file, giving all parameters for training up a Weight list
for landmark.,
flag: --inputTrainingList %s
outputWeightsList: (a boolean or a file name)
, The filename of a csv file which is a list of landmarks and their
corresponding weights.,
flag: --outputWeightsList %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputWeightsList: (an existing file name)
, The filename of a csv file which is a list of landmarks and their
corresponding weights.,
```


interfaces.slicer.base

SlicerCommandLine

[Link to code](#)

Wraps command **None**

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nitype default value: {})
    Environment variables
ignore_exception: (a boolean, nitype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
None
```

interfaces.slicer.converters

DicomToNrrdConverter

[Link to code](#)

Wraps command ****DicomToNrrdConverter****

    : DICOM to NRRD Converter

category: Converters

description: Converts diffusion weighted MR images in dicom series into Nrrd format for analysis in Slicer. This program has been tested on only a limited subset of DTI dicom formats available from Siemens, GE, and Phillips scanners. Work in progress to support dicom multi-frame data. The program parses dicom header to extract necessary information about measurement frame, diffusion weighting directions, b-values, etc, and write out a nrrd image. For non-diffusion weighted dicom images, it loads in an entire dicom series and writes out a single dicom volume in a .nhdr/.raw pair.

version: 0.2.0.\$Revision: 916 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/DicomToNrrdConverter>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Xiaodong Tao (GE), Vince Magnotta (UIowa), Hans Johnson (UIowa)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Additional support for DTI data produced on Philips scanners was contributed by Vincent Magnotta and Hans Johnson at the University of Iowa.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputDicomDirectory: (an existing directory name)
    Directory holding Dicom series
    flag: --inputDicomDirectory %s
outputDirectory: (a boolean or a directory name)
    Directory holding the output NRRD format
    flag: --outputDirectory %s
outputVolume: (a unicode string)
    Output filename (.nhdr or .nrrd)
    flag: --outputVolume %s
smallGradientThreshold: (a float)
    If a gradient magnitude is greater than 0 and less than
    smallGradientThreshold, then DicomToNrrdConverter will display an
    error message and quit, unless the useBMatrixGradientDirections
    option is set.
    flag: --smallGradientThreshold %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
useBMatrixGradientDirections: (a boolean)
    Fill the nhdr header with the gradient directions and bvalues
    computed out of the BMatrix. Only changes behavior for Siemens data.
    flag: --useBMatrixGradientDirections
useIdentityMeaseurementFrame: (a boolean)
    Adjust all the gradients so that the measurement frame is an
```

```

identity matrix.
flag: --useIdentityMeasurementFrame
writeProtocolGradientsFile: (a boolean)
    Write the protocol gradients to a file suffixed by '.txt' as they
    were specified in the protocol by multiplying each diffusion gradient
    direction by the measurement frame. This file is for debugging
    purposes only, the format is not fixed, and will likely change as
    debugging of new dicom formats is necessary.
flag: --writeProtocolGradientsFile

```

Outputs:

```

outputDirectory: (an existing directory name)
    Directory holding the output NRRD format

```

OrientScalarVolume

[Link to code](#)Wraps command ****OrientScalarVolume****

title: Orient Scalar Volume

category: Converters

description: Orients an output volume. Rearranges the slices in a volume according to the selected orientation. The slices are not interpolated. They are just reordered and/or permuted. The resulting volume will cover the original volume. NOTE: since Slicer takes into account the orientation of a volume, the re-oriented volume will not show any difference from the original volume, To see the difference, save the volume and display it with a system that either ignores the orientation of the image (e.g. Paraview) or displays individual images.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/OrientImage>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume1: (an existing file name)
    Input volume 1
    flag: %s, position: -2
orientation: ('Axial' or 'Coronal' or 'Sagittal' or 'RIP' or 'LIP' or
    'RSP' or 'LSP' or 'RIA' or 'LIA' or 'RSA' or 'LSA' or 'IRP' or
    'ILP' or 'SRP' or 'SLP' or 'IRA' or 'ILA' or 'SRA' or 'SLA' or
    'RPI' or 'LPI' or 'RAI' or 'LAI' or 'RPS' or 'LPS' or 'RAS' or
    'LAS' or 'PRI' or 'PLI' or 'ARI' or 'ALI' or 'PRS' or 'PLS' or
    'ARS' or 'ALS' or 'IPR' or 'SPR' or 'IAR' or 'SAR' or 'IPL' or
    'SPL' or 'IAL' or 'SAL' or 'PIR' or 'PSR' or 'AIR' or 'ASR' or
    'PIL' or 'PSL' or 'AIL' or 'ASL')

```

```

Orientation choices
flag: --orientation %s
outputVolume: (a boolean or a file name)
    The oriented volume
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    The oriented volume

```

interfaces.slicer.diffusion.diffusion

DTIexport

[Link to code](#)Wraps command ****DTIexport****

title: DTIexport

category: Diffusion.Diffusion Data Conversion

description: Export DTI data to various file formats

version: 1.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/DTIExport>

contributor: Sonia Pujol (SPL, BWH)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputTensor: (an existing file name)
    Input DTI volume
    flag: %s, position: -2
outputFile: (a boolean or a file name)
    Output DTI file
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```
outputFile: (an existing file name)
            Output DTI file
```

DTIimport

[Link to code](#)

Wraps command ****DTIimport****

title: DTIimport

category: Diffusion.Diffusion Data Conversion

description: Import tensor datasets from various formats, including the NifTi file format

version: 1.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/DTIImport>

contributor: Sonia Pujol (SPL, BWH)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NA-MIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
          Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
inputFile: (an existing file name)
            Input DTI file
            flag: %s, position: -2
outputTensor: (a boolean or a file name)
              Output DTI volume
              flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
testingmode: (a boolean)
              Enable testing mode. Sample helix file (helix-DTI.nhdr) will be
              loaded into Slicer and converted in Nifti.
              flag: --testingmode
```

Outputs:

```
outputTensor: (an existing file name)
              Output DTI volume
```

DWJointRicianLMMSEFilter

[Link to code](#)

Wraps command ****DWJointRicianLMMSEFilter****

title: DWI Joint Rician LMMSE Filter

category: Diffusion.Diffusion Weighted Images

description: This module reduces Rician noise (or unwanted detail) on a set of diffusion weighted images. For this, it filters the image in the mean squared error sense using a Rician noise model. The N closest gradient directions to the direction being processed are filtered together to improve the results: the noise-free signal is seen as an n-dimensional vector which has to be estimated with the LMMSE method from a set of corrupted measurements. To that end, the covariance matrix of the noise-free vector and the cross covariance between this signal and the noise have to be estimated, which is done taking into account the image formation process. The noise parameter is automatically estimated from a rough segmentation of the background of the image. In this area the signal is simply 0, so that Rician statistics reduce to Rayleigh and the noise power can be easily estimated from the mode of the histogram. A complete description of the algorithm may be found in: Antonio Tristan-Vega and Santiago Aja-Fernandez, DWI filtering using joint information for DTI and HARDI, Medical Image Analysis, Volume 14, Issue 2, Pages 205-218. 2010.

version: 0.1.1.\$Revision: 1 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/JointRicianLMMSEImageFilter>

contributor: Antonio Tristan Vega (UVa), Santiago Aja Fernandez (UVa)

acknowledgements: Partially founded by grant number TEC2007-67073/TCM from the Comision Interministerial de Ciencia y Tecnologia (Spain).

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
compressOutput: (a boolean)
    Compress the data of the compressed file using gzip
    flag: --compressOutput
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input DWI volume.
    flag: %s, position: -2
ng: (an integer (int or long))
    The number of the closest gradients that are used to jointly filter
    a given gradient direction (0 to use all).
    flag: --ng %d
outputVolume: (a boolean or a file name)
    Output DWI volume.
    flag: %s, position: -1
re: (a list of items which are an integer (int or long))
    Estimation radius.
    flag: --re %s
rf: (a list of items which are an integer (int or long))
    Filtering radius.
    flag: --rf %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Output DWI volume.
```

DWIRicianLMMSEFilter

[Link to code](#)

Wraps command ****DWIRicianLMMSEFilter****

title: DWI Rician LMMSE Filter

category: Diffusion.Diffusion Weighted Images

description: This module reduces noise (or unwanted detail) on a set of diffusion weighted images. For this, it filters the image in the mean squared error sense using a Rician noise model. Images corresponding to each gradient direction, including baseline, are processed individually. The noise parameter is automatically estimated (noise estimation improved but slower). Note that this is a general purpose filter for MRi images. The module jointLMMSE has been specifically designed for DWI volumes and shows a better performance, so its use is recommended instead. A complete description of the algorithm in this module can be found in: S. Aja-Fernandez, M. Niethammer, M. Kubicki, M. Shenton, and C.-F. Westin. Restoration of DWI data using a Rician LMMSE estimator. IEEE Transactions on Medical Imaging, 27(10): pp. 1389-1403, Oct. 2008.

version: 0.1.1.\$Revision: 1 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/RicianLMMSEImageFilter>

contributor: Antonio Tristan Vega (UVa), Santiago Aja Fernandez (UVa), Marc Niethammer (UNC)

acknowledgements: Partially founded by grant number TEC2007-67073/TCM from the Comision Interministerial de Ciencia y Tecnologia (Spain).

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
compressOutput: (a boolean)
    Compress the data of the compressed file using gzip
    flag: --compressOutput
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
hrf: (a float)
    How many histogram bins per unit interval.
    flag: --hrf %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input DWI volume.
    flag: %s, position: -2
iter: (an integer (int or long))
    Number of iterations for the noise removal filter.
    flag: --iter %d
maxnstd: (an integer (int or long))
    Maximum allowed noise standard deviation.
    flag: --maxnstd %d
minnstd: (an integer (int or long))
    Minimum allowed noise standard deviation.
    flag: --minnstd %d
```

```

mnve: (an integer (int or long))
    Minimum number of voxels in kernel used for estimation.
    flag: --mnve %d
mnvf: (an integer (int or long))
    Minimum number of voxels in kernel used for filtering.
    flag: --mnvf %d
outputVolume: (a boolean or a file name)
    Output DWI volume.
    flag: %s, position: -1
re: (a list of items which are an integer (int or long))
    Estimation radius.
    flag: --re %s
rf: (a list of items which are an integer (int or long))
    Filtering radius.
    flag: --rf %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
uav: (a boolean)
    Use absolute value in case of negative square.
    flag: --uav

```

Outputs:

```

outputVolume: (an existing file name)
    Output DWI volume.

```

DWIToDTIEstimation[Link to code](#)Wraps command ****DWIToDTIEstimation****

title: DWI to DTI Estimation

category: Diffusion.Diffusion Weighted Images

description: Performs a tensor model estimation from diffusion weighted images.

There are three estimation methods available: least squares, weighed least squares and non-linear estimation. The first method is the traditional method for tensor estimation and the fastest one. Weighted least squares takes into account the noise characteristics of the MRI images to weight the DWI samples used in the estimation based on its intensity magnitude. The last method is the more complex.

version: 0.1.0.\$Revision: 1892 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/DiffusionTensorEstimation>

license: slicer3

contributor: Raul San Jose (SPL, BWH)

acknowledgements: This command module is based on the estimation functionality provided by the Teem library. This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
enumeration: ('LS' or 'WLS')

```



```

    LS: Least Squares, WLS: Weighted Least Squares
    flag: --enumeration %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input DWI volume
    flag: %s, position: -3
mask: (an existing file name)
    Mask where the tensors will be computed
    flag: --mask %s
outputBaseline: (a boolean or a file name)
    Estimated baseline volume
    flag: %s, position: -1
outputTensor: (a boolean or a file name)
    Estimated DTI volume
    flag: %s, position: -2
shiftNeg: (a boolean)
    Shift eigenvalues so all are positive (accounts for bad tensors
    related to noise or acquisition error)
    flag: --shiftNeg
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputBaseline: (an existing file name)
    Estimated baseline volume
outputTensor: (an existing file name)
    Estimated DTI volume

```

DiffusionTensorScalarMeasurements

Link to code

Wraps command ****DiffusionTensorScalarMeasurements****

title: Diffusion Tensor Scalar Measurements

category: Diffusion.Diffusion Tensor Images

description: Compute a set of different scalar measurements from a tensor field, specially oriented for Diffusion Tensors where some rotationally invariant measurements, like Fractional Anisotropy, are highly used to describe the anisotropic behaviour of the tensor.

version: 0.1.0.\$Revision: 1892 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/DiffusionTensorMathematics>

contributor: Raul San Jose (SPL, BWH)

acknowledgements: LMI

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)

```

```

    Additional parameters to the command
    flag: %s
enumeration: ('Trace' or 'Determinant' or 'RelativeAnisotropy' or
              'FractionalAnisotropy' or 'Mode' or 'LinearMeasure' or
              'PlanarMeasure' or 'SphericalMeasure' or 'MinEigenvalue' or
              'MidEigenvalue' or 'MaxEigenvalue' or 'MaxEigenvalueProjectionX' or
              'MaxEigenvalueProjectionY' or 'MaxEigenvalueProjectionZ' or
              'RAIMaxEigenvecX' or 'RAIMaxEigenvecY' or 'RAIMaxEigenvecZ' or
              'MaxEigenvecX' or 'MaxEigenvecY' or 'MaxEigenvecZ' or 'D11' or
              'D22' or 'D33' or 'ParallelDiffusivity' or
              'PerpendicularDffusivity')
An enumeration of strings
flag: --enumeration %s
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a value
          of class 'str', nipy default value: {})
Environment variables
ignore_exception: (a boolean, nipy default value: False)
Print an error message instead of throwing an exception in case the
interface fails to run
inputVolume: (an existing file name)
Input DTI volume
flag: %s, position: -3
outputScalar: (a boolean or a file name)
Scalar volume derived from tensor
flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputScalar: (an existing file name)
Scalar volume derived from tensor

```

DiffusionWeightedVolumeMasking[Link to code](#)Wraps command ****DiffusionWeightedVolumeMasking****

title: Diffusion Weighted Volume Masking

category: Diffusion.Diffusion Weighted Images

description:

Performs a mask calculation from a diffusion weighted (DW) image.

Starting from a dw image, this module computes the baseline image averaging all the images without diffusion weighting and then applies the otsu segmentation algorithm in order to produce a mask. this mask can then be used when estimating the diffusion tensor (dt) image, not to estimate tensors all over the volume.

version: 0.1.0.\$Revision: 1892 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/DiffusionWeightedMasking>

license: slicer3

contributor: Demian Wassermann (SPL, BWH)

Inputs:

[Mandatory]

[Optional]

args: (a unicode string)

```

    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input DWI volume
    flag: %s, position: -4
otsuomegathreshold: (a float)
    Control the sharpness of the threshold in the Otsu computation. 0:
    lower threshold, 1: higher threshold
    flag: --otsuomegathreshold %f
outputBaseline: (a boolean or a file name)
    Estimated baseline volume
    flag: %s, position: -2
removeislands: (a boolean)
    Remove Islands in Threshold Mask?
    flag: --removeislands
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thresholdMask: (a boolean or a file name)
    Otsu Threshold Mask
    flag: %s, position: -1

```

Outputs:

```

outputBaseline: (an existing file name)
    Estimated baseline volume
thresholdMask: (an existing file name)
    Otsu Threshold Mask

```

ResampleDTIVolume[Link to code](#)Wraps command ****ResampleDTIVolume****

title: Resample DTI Volume

category: Diffusion.Diffusion Tensor Images

description: Resampling an image is a very important task in image analysis. It is especially important in the frame of image registration. This module implements DT image resampling through the use of itk Transforms. The resampling is controlled by the Output Spacing. “Resampling” is performed in space coordinates, not pixel/grid coordinates. It is quite important to ensure that image spacing is properly set on the images involved. The interpolator is required since the mapping from one space to the other will often require evaluation of the intensity of the image at non-grid positions.

version: 0.1

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ResampleDTI>

contributor: Francois Budin (UNC)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from <http://nihroadmap.nih.gov/bioinformatics>

Inputs:

```

[Mandatory]

[Optional]
Inverse_ITK_Transformation: (a boolean)
    Inverse the transformation before applying it from output image to
    input image (only for rigid and affine transforms)
    flag: --Inverse_ITK_Transformation
Reference: (an existing file name)
    Reference Volume (spacing,size,orientation,origin)
    flag: --Reference %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
centered_transform: (a boolean)
    Set the center of the transformation to the center of the input
    image (only for rigid and affine transforms)
    flag: --centered_transform
correction: ('zero' or 'none' or 'abs' or 'nearest')
    Correct the tensors if computed tensor is not semi-definite positive
    flag: --correction %s
defField: (an existing file name)
    File containing the deformation field (3D vector image containing
    vectors with 3 components)
    flag: --defField %s
default_pixel_value: (a float)
    Default pixel value for samples falling outside of the input region
    flag: --default_pixel_value %f
direction_matrix: (a list of items which are a float)
    9 parameters of the direction matrix by rows (ijk to LPS if LPS
    transform, ijk to RAS if RAS transform)
    flag: --direction_matrix %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
hfieldtype: ('displacement' or 'h-Field')
    Set if the deformation field is an -Field
    flag: --hfieldtype %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
image_center: ('input' or 'output')
    Image to use to center the transform (used only if 'Centered
    Transform' is selected)
    flag: --image_center %s
inputVolume: (an existing file name)
    Input volume to be resampled
    flag: %s, position: -2
interpolation: ('linear' or 'nn' or 'ws' or 'bs')
    Sampling algorithm (linear , nn (nearest neighborhood), ws
    (WindowedSinc), bs (BSpline) )
    flag: --interpolation %s
notbulk: (a boolean)
    The transform following the BSpline transform is not set as a bulk
    transform for the BSpline transform
    flag: --notbulk
number_of_thread: (an integer (int or long))
    Number of thread used to compute the output image

```

```

    flag: --number_of_thread %d
origin: (a list of items which are any value)
    Origin of the output Image
    flag: --origin %s
outputVolume: (a boolean or a file name)
    Resampled Volume
    flag: %s, position: -1
rotation_point: (a list of items which are any value)
    Center of rotation (only for rigid and affine transforms)
    flag: --rotation_point %s
size: (a list of items which are a float)
    Size along each dimension (0 means use input size)
    flag: --size %s
spaceChange: (a boolean)
    Space Orientation between transform and image is different (RAS/LPS)
    (warning: if the transform is a Transform Node in Slicer3, do not
    select)
    flag: --spaceChange
spacing: (a list of items which are a float)
    Spacing along each dimension (0 means use input spacing)
    flag: --spacing %s
spline_order: (an integer (int or long))
    Spline Order (Spline order may be from 0 to 5)
    flag: --spline_order %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transform: ('rt' or 'a')
    Transform algorithm, rt = Rigid Transform, a = Affine Transform
    flag: --transform %s
transform_matrix: (a list of items which are a float)
    12 parameters of the transform matrix by rows ( --last 3 being
    translation-- )
    flag: --transform_matrix %s
transform_order: ('input-to-output' or 'output-to-input')
    Select in what order the transforms are read
    flag: --transform_order %s
transform_tensor_method: ('PPD' or 'FS')
    Chooses between 2 methods to transform the tensors: Finite Strain
    (FS), faster but less accurate, or Preservation of the Principal
    Direction (PPD)
    flag: --transform_tensor_method %s
transformationFile: (an existing file name)
    flag: --transformationFile %s
window_function: ('h' or 'c' or 'w' or 'l' or 'b')
    Window Function , h = Hamming , c = Cosine , w = Welch , l = Lanczos
    , b = Blackman
    flag: --window_function %s

```

Outputs:

```

outputVolume: (an existing file name)
    Resampled Volume

```

TractographyLabelMapSeeding[Link to code](#)

Wraps command ****TractographyLabelMapSeeding****

title: Tractography Label Map Seeding

category: Diffusion.Diffusion Tensor Images

description: Seed tracts on a Diffusion Tensor Image (DT) from a label map

version: 0.1.0.\$Revision: 1892 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/Seeding>

license: slicer3

contributor: Raul San Jose (SPL, BWH), Demian Wassermann (SPL, BWH)

acknowledgements: Laboratory of Mathematics in Imaging. This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
InputVolume: (an existing file name)
    Input DTI volume
    flag: %s, position: -2
OutputFibers: (a boolean or a file name)
    Tractography result
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
clthreshold: (a float)
    Minimum Linear Measure for the seeding to start.
    flag: --clthreshold %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputroi: (an existing file name)
    Label map with seeding ROIs
    flag: --inputroi %s
integrationsteplength: (a float)
    Distance between points on the same fiber in mm
    flag: --integrationsteplength %f
label: (an integer (int or long))
    Label value that defines seeding region.
    flag: --label %d
maximumlength: (a float)
    Maximum length of fibers (in mm)
    flag: --maximumlength %f
minimumlength: (a float)
    Minimum length of the fibers (in mm)
    flag: --minimumlength %f
name: (a unicode string)
    Name to use for fiber files
    flag: --name %s
outputdirectory: (a boolean or a directory name)
    Directory in which to save fiber(s)
    flag: --outputdirectory %s
randomgrid: (a boolean)
    Enable random placing of seeds
```

```

    flag: --randomgrid
seedspace: (a float)
    Spacing (in mm) between seed points, only matters if use Use Index
    Space is off
    flag: --seedspace %f
stoppingcurvature: (a float)
    Tractography will stop if radius of curvature becomes smaller than
    this number units are degrees per mm
    flag: --stoppingcurvature %f
stoppingmode: ('LinearMeasure' or 'FractionalAnisotropy')
    Tensor measurement used to stop the tractography
    flag: --stoppingmode %s
stoppingvalue: (a float)
    Tractography will stop when the stopping measurement drops below
    this value
    flag: --stoppingvalue %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
useindexspace: (a boolean)
    Seed at IJK voxel grid
    flag: --useindexspace
writetofile: (a boolean)
    Write fibers to disk or create in the scene?
    flag: --writetofile

```

Outputs:

```

OutputFibers: (an existing file name)
    Tractography result
outputdirectory: (an existing directory name)
    Directory in which to save fiber(s)

```

interfaces.slicer.filtering.arithmetic**AddScalarVolumes**[Link to code](#)Wraps command ****AddScalarVolumes****

title: Add Scalar Volumes

category: Filtering.Arithmetic

description: Adds two images. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/Add>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command

```

```

    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume1: (an existing file name)
    Input volume 1
    flag: %s, position: -3
inputVolume2: (an existing file name)
    Input volume 2
    flag: %s, position: -2
order: ('0' or '1' or '2' or '3')
    Interpolation order if two images are in different coordinate frames
    or have different sampling.
    flag: --order %s
outputVolume: (a boolean or a file name)
    Volume1 + Volume2
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Volume1 + Volume2

```

CastScalarVolume[Link to code](#)Wraps command ****CastScalarVolume ****

title: Cast Scalar Volume

category: Filtering.Arithmetic

description: Cast a volume to a given data type. Use at your own risk when casting an input volume into a lower precision type! Allows casting to the same type as the input volume.

version: 0.1.0.\$Revision: 2104 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/Cast>

contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
InputVolume: (an existing file name)
    Input volume, the volume to cast.
    flag: %s, position: -2
OutputVolume: (a boolean or a file name)
    Output volume, cast to the new type.
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command

```



```

    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
type: ('Char' or 'UnsignedChar' or 'Short' or 'UnsignedShort' or
    'Int' or 'UnsignedInt' or 'Float' or 'Double')
    Type for the new output volume.
flag: --type %s

```

Outputs:

```

OutputVolume: (an existing file name)
    Output volume, cast to the new type.

```

MaskScalarVolume[Link to code](#)Wraps command ****MaskScalarVolume****

title: Mask Scalar Volume

category: Filtering.Arithmetic

description: Masks two images. The output image is set to 0 everywhere except where the chosen label from the mask volume is present, at which point it will retain it's original values. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.

version: 0.1.0.\$Revision: 8595 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/Mask>

contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
InputVolume: (an existing file name)
    Input volume to be masked
    flag: %s, position: -3
MaskVolume: (an existing file name)
    Label volume containing the mask
    flag: %s, position: -2
OutputVolume: (a boolean or a file name)
    Output volume: Input Volume masked by label value from Mask Volume
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
label: (an integer (int or long))
    Label value in the Mask Volume to use as the mask
    flag: --label %d
replace: (an integer (int or long))
    Value to use for the output volume outside of the mask
    flag: --replace %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

OutputVolume: (an existing file name)
    Output volume: Input Volume masked by label value from Mask Volume

```

MultiplyScalarVolumes[Link to code](#)Wraps command ****MultiplyScalarVolumes****

title: Multiply Scalar Volumes

category: Filtering.Arithmetic

description: Multiplies two images. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.

version: 0.1.0.\$Revision: 8595 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/Multiply>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume1: (an existing file name)
    Input volume 1
    flag: %s, position: -3
inputVolume2: (an existing file name)
    Input volume 2
    flag: %s, position: -2
order: ('0' or '1' or '2' or '3')
    Interpolation order if two images are in different coordinate frames
    or have different sampling.
    flag: --order %s

```

```

outputVolume: (a boolean or a file name)
    Volume1 * Volume2
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Volume1 * Volume2

```

SubtractScalarVolumes[Link to code](#)

Wraps command ****SubtractScalarVolumes****

title: Subtract Scalar Volumes

category: Filtering.Arithmetic

description: Subtracts two images. Although all image types are supported on input, only signed types are produced. The two images do not have to have the same dimensions.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/Subtract>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume1: (an existing file name)
    Input volume 1
    flag: %s, position: -3
inputVolume2: (an existing file name)
    Input volume 2
    flag: %s, position: -2
order: ('0' or '1' or '2' or '3')
    Interpolation order if two images are in different coordinate frames
    or have different sampling.
    flag: --order %s
outputVolume: (a boolean or a file name)
    Volume1 - Volume2
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display

```

```
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
               Volume1 - Volume2
```

interfaces.slicer.filtering.checkerboardfilter

CheckerBoardFilter

[Link to code](#)Wraps command ****CheckerBoardFilter****

title: CheckerBoard Filter

category: Filtering

description: Create a checkerboard volume of two volumes. The output volume will show the two inputs alternating according to the user supplied checkerPattern. This filter is often used to compare the results of image registration. Note that the second input is resampled to the same origin, spacing and direction before it is composed with the first input. The scalar type of the output volume will be the same as the input image scalar type.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/CheckerBoard>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
checkerPattern: (a list of items which are an integer (int or long))
      The pattern of input 1 and input 2 in the output image. The user can
      specify the number of checkers in each dimension. A checkerPattern
      of 2,2,1 means that images will alternate in every other checker in
      the first two dimensions. The same pattern will be used in the 3rd
      dimension.
      flag: --checkerPattern %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
      Print an error message instead of throwing an exception in case the
      interface fails to run
inputVolume1: (an existing file name)
      First Input volume
      flag: %s, position: -3
inputVolume2: (an existing file name)
      Second Input volume
      flag: %s, position: -2
outputVolume: (a boolean or a file name)
      Output filtered
      flag: %s, position: -1
```

```
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Output filtered
```

interfaces.slicer.filtering.denoising

CurvatureAnisotropicDiffusion

[Link to code](#)

Wraps command ****CurvatureAnisotropicDiffusion****

title: Curvature Anisotropic Diffusion

category: Filtering.Denoising

description: Performs anisotropic diffusion on an image using a modified curvature diffusion equation (MCDE). MCDE does not exhibit the edge enhancing properties of classic anisotropic diffusion, which can under certain conditions undergo a ‘negative’ diffusion, which enhances the contrast of edges. Equations of the form of MCDE always undergo positive diffusion, with the conductance term only varying the strength of that diffusion.

Qualitatively, MCDE compares well with other non-linear diffusion techniques. It is less sensitive to contrast than classic Perona-Malik style diffusion, and preserves finer detailed structures in images. There is a potential speed trade-off for using this function in place of Gradient Anisotropic Diffusion. Each iteration of the solution takes roughly twice as long. Fewer iterations, however, may be required to reach an acceptable solution.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/CurvatureAnisotropicDiffusion>

contributor: Bill Lorensen (GE)

acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software Consortium

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
conductance: (a float)
    Conductance controls the sensitivity of the conductance term. As a
    general rule, the lower the value, the more strongly the filter
    preserves edges. A high value will cause diffusion (smoothing)
    across edges. Note that the number of iterations controls how much
    smoothing is done within regions bounded by edges.
    flag: --conductance %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

```

inputVolume: (an existing file name)
    Input volume to be filtered
    flag: %s, position: -2
iterations: (an integer (int or long))
    The more iterations, the more smoothing. Each iteration takes the
    same amount of time. If it takes 10 seconds for one iteration, then
    it will take 100 seconds for 10 iterations. Note that the
    conductance controls how much each iteration smooths across edges.
    flag: --iterations %d
outputVolume: (a boolean or a file name)
    Output filtered
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timeStep: (a float)
    The time step depends on the dimensionality of the image. In Slicer
    the images are 3D and the default (.0625) time step will provide a
    stable solution.
    flag: --timeStep %f

```

Outputs:

```

outputVolume: (an existing file name)
    Output filtered

```

GaussianBlurImageFilter[Link to code](#)Wraps command ****GaussianBlurImageFilter****

title: Gaussian Blur Image Filter

category: Filtering.Denoising

description: Apply a gaussian blurr to an image

version: 0.1.0.\$Revision: 1.1 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/GaussianBlurImageFilter>

contributor: Julien Jomier (Kitware), Stephen Aylward (Kitware)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume

```

```

    flag: %s, position: -2
outputVolume: (a boolean or a file name)
    Blurred Volume
    flag: %s, position: -1
sigma: (a float)
    Sigma value in physical units (e.g., mm) of the Gaussian kernel
    flag: --sigma %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Blurred Volume

```

GradientAnisotropicDiffusion[Link to code](#)Wraps command ****GradientAnisotropicDiffusion ****

title: Gradient Anisotropic Diffusion

category: Filtering.Denoising

description: Runs gradient anisotropic diffusion on a volume.

Anisotropic diffusion methods reduce noise (or unwanted detail) in images while preserving specific image features, like edges. For many applications, there is an assumption that light-dark transitions (edges) are interesting. Standard isotropic diffusion methods move and blur light-dark boundaries. Anisotropic diffusion methods are formulated to specifically preserve edges. The conductance term for this implementation is a function of the gradient magnitude of the image at each point, reducing the strength of diffusion at edges. The numerical implementation of this equation is similar to that described in the Perona-Malik paper, but uses a more robust technique for gradient magnitude estimation and has been generalized to N-dimensions.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: [http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/](http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/GradientAnisotropicDiffusion)[GradientAnisotropicDiffusion](#)

contributor: Bill Lorensen (GE)

acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software Consortium

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
conductance: (a float)
    Conductance controls the sensitivity of the conductance term. As a
    general rule, the lower the value, the more strongly the filter
    preserves edges. A high value will cause diffusion (smoothing)
    across edges. Note that the number of iterations controls how much
    smoothing is done within regions bounded by edges.
    flag: --conductance %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume to be filtered
    flag: %s, position: -2
iterations: (an integer (int or long))
    The more iterations, the more smoothing. Each iteration takes the
    same amount of time. If it takes 10 seconds for one iteration, then
    it will take 100 seconds for 10 iterations. Note that the
    conductance controls how much each iteration smooths across edges.
    flag: --iterations %d
outputVolume: (a boolean or a file name)
    Output filtered
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timeStep: (a float)
    The time step depends on the dimensionality of the image. In Slicer
    the images are 3D and the default (.0625) time step will provide a
    stable solution.
    flag: --timeStep %f

```

Outputs:

```

outputVolume: (an existing file name)
    Output filtered

```

MedianImageFilter[Link to code](#)Wraps command ****MedianImageFilter****

title: Median Image Filter

category: Filtering.Denoising

description: The MedianImageFilter is commonly used as a robust approach for noise reduction. This filter is particularly efficient against “salt-and-pepper” noise. In other words, it is robust to the presence of gray-level outliers. MedianImageFilter computes the value of each output pixel as the statistical median of the neighborhood of values around the corresponding input pixel.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/MedianImageFilter>

contributor: Bill Lorensen (GE)

acknowledgements: This command module was derived from Insight/Examples/Filtering/MedianImageFilter (copyright) Insight Software Consortium

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```



```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume to be filtered
    flag: %s, position: -2
neighborhood: (a list of items which are an integer (int or long))
    The size of the neighborhood in each dimension
    flag: --neighborhood %s
outputVolume: (a boolean or a file name)
    Output filtered
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Output filtered

```

interfaces.slicer.filtering.extractskel­eton

ExtractSkeleton

[Link to code](#)Wraps command ****ExtractSkeleton****

title: Extract Skeleton

category: Filtering

description: Extract the skeleton of a binary object. The skeleton can be limited to being a 1D curve or allowed to be a full 2D manifold. The branches of the skeleton can be pruned so that only the maximal center skeleton is returned.

version: 0.1.0.\$Revision: 2104 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ExtractSkeleton>

contributor: Pierre Seroul (UNC), Martin Styner (UNC), Guido Gerig (UNC), Stephen Aylward (Kitware)

acknowledgements: The original implementation of this method was provided by ETH Zurich, Image Analysis Laboratory of Profs Olaf Kuebler, Gabor Szekely and Guido Gerig. Martin Styner at UNC, Chapel Hill made enhancements. Wrapping for Slicer was provided by Pierre Seroul and Stephen Aylward at Kitware, Inc.

Inputs:

```

[Mandatory]

[Optional]
InputImageFileName: (an existing file name)
    Input image
    flag: %s, position: -2
OutputImageFileName: (a boolean or a file name)
    Skeleton of the input image
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
dontPrune: (a boolean)
    Return the full skeleton, not just the maximal skeleton
    flag: --dontPrune

```

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
         Print an error message instead of throwing an exception in case the
         interface fails to run
numPoints: (an integer (int or long))
         Number of points used to represent the skeleton
         flag: --numPoints %d
pointsFile: (a unicode string)
         Name of the file to store the coordinates of the central (1D)
         skeleton points
         flag: --pointsFile %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
         Control terminal output: `stream` - displays to terminal immediately
         (default), `allatonce` - waits till command is finished to display
         output, `file` - writes output to file, `none` - output is ignored
type: ('1D' or '2D')
         Type of skeleton to create
         flag: --type %s

```

Outputs:

```

OutputImageFileName: (an existing file name)
         Skeleton of the input image

```

interfaces.slicer.filtering.histogrammatching**HistogramMatching**[Link to code](#)Wraps command ****HistogramMatching ****

title: Histogram Matching

category: Filtering

description: Normalizes the grayscale values of a source image based on the grayscale values of a reference image. This filter uses a histogram matching technique where the histograms of the two images are matched only at a specified number of quantile values.

The filter was originally designed to normalize MR images of the sameMR protocol and same body part. The algorithm works best if background pixels are excluded from both the source and reference histograms. A simple background exclusion method is to exclude all pixels whose grayscale values are smaller than the mean grayscale value. ThresholdAtMeanIntensity switches on this simple background exclusion method.

Number of match points governs the number of quantile values to be matched.

The filter assumes that both the source and reference are of the same type and that the input and output image type have the same number of dimension and have scalar pixel types.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/HistogramMatching>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]
```

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyre default value: {})
    Environment variables
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume to be filtered
    flag: %s, position: -3
numberOfHistogramLevels: (an integer (int or long))
    The number of hisogram levels to use
    flag: --numberOfHistogramLevels %d
numberOfMatchPoints: (an integer (int or long))
    The number of match points to use
    flag: --numberOfMatchPoints %d
outputVolume: (a boolean or a file name)
    Output volume. This is the input volume with intensities matched to
    the reference volume.
    flag: %s, position: -1
referenceVolume: (an existing file name)
    Input volume whose histogram will be matched
    flag: %s, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (a boolean)
    If on, only pixels above the mean in each volume are thresholded.
    flag: --threshold
```

Outputs:

```
outputVolume: (an existing file name)
    Output volume. This is the input volume with intensities matched to
    the reference volume.
```

interfaces.slicer.filtering.imagelabelcombine

ImageLabelCombine

[Link to code](#)

Wraps command ****ImageLabelCombine****

title: Image Label Combine

category: Filtering

description: Combine two label maps into one

version: 0.1.0

documentation-url: [http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/](http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ImageLabelCombine)

[ImageLabelCombine](#)

contributor: Alex Yarmarkovich (SPL, BWH)

Inputs:

```

[Mandatory]

[Optional]
InputLabelMap_A: (an existing file name)
    Label map image
    flag: %s, position: -3
InputLabelMap_B: (an existing file name)
    Label map image
    flag: %s, position: -2
OutputLabelMap: (a boolean or a file name)
    Resulting Label map image
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
first_overwrites: (a boolean)
    Use first or second label when both are present
    flag: --first_overwrites
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

OutputLabelMap: (an existing file name)
    Resulting Label map image

```

interfaces.slicer.filtering.morphology

GrayscaleFillHoleImageFilter

[Link to code](#)Wraps command ****GrayscaleFillHoleImageFilter****

title: Grayscale Fill Hole Image Filter

category: Filtering.Morphology

description: GrayscaleFillholeImageFilter fills holes in a grayscale image. Holes are local minima in the grayscale topography that are not connected to boundaries of the image. Gray level values adjacent to a hole are extrapolated across the hole.

This filter is used to smooth over local minima without affecting the values of local maxima. If you take the difference between the output of this filter and the original image (and perhaps threshold the difference above a small value), you'll obtain a map of the local minima.

This filter uses the itkGrayscaleGeodesicErodeImageFilter. It provides its own input as the “mask” input to the geodesic erosion. The “marker” image for the geodesic erosion is constructed such that boundary pixels match the boundary pixels of the input image and the interior pixels are set to the maximum pixel value in the input image.

Geodesic morphology and the Fillhole algorithm is described in Chapter 6 of Pierre Soille’s book “Morphological Image Analysis: Principles and Applications”, Second Edition, Springer, 2003.

A companion filter, Grayscale Grind Peak, removes peaks in grayscale images.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documenta­tion-url: <http://wiki.slicer.org/slicerWiki/index.php/Documenta­tion/4.1/Modules/GrayscaleFillHoleImageFilter>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume to be filtered
    flag: %s, position: -2
outputVolume: (a boolean or a file name)
    Output filtered
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Output filtered
```

GrayscaleGrindPeakImageFilter

[Link to code](#)

Wraps command ****GrayscaleGrindPeakImageFilter****

title: Grayscale Grind Peak Image Filter

category: Filtering.Morphology

description: GrayscaleGrindPeakImageFilter removes peaks in a grayscale image. Peaks are local maxima in the grayscale topography that are not connected to boundaries of the image. Gray level values adjacent to a peak are extrapolated through the peak.

This filter is used to smooth over local maxima without affecting the values of local minima. If you take the difference between the output of this filter and the original image (and perhaps threshold the difference above a small value), you'll obtain a map of the local maxima.

This filter uses the GrayscaleGeodesicDilateImageFilter. It provides its own input as the “mask” input to the geodesic erosion. The “marker” image for the geodesic erosion is constructed such that boundary pixels match the boundary pixels of the input image and the interior pixels are set to the minimum pixel value in the input image.

This filter is the dual to the GrayscaleFillholeImageFilter which implements the Fillhole algorithm. Since it is a dual, it is somewhat superfluous but is provided as a convenience.

Geodesic morphology and the Fillhole algorithm is described in Chapter 6 of Pierre Soille’s book “Morphological Image Analysis: Principles and Applications”, Second Edition, Springer, 2003.

A companion filter, Grayscale Fill Hole, fills holes in grayscale images.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/GrayscaleGrindPeakImageFilter>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume to be filtered
    flag: %s, position: -2
outputVolume: (a boolean or a file name)
    Output filtered
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
    Output filtered
```

interfaces.slicer.filtering.n4itkbiasfieldcorrection

N4ITKBiasFieldCorrection

[Link to code](#)

Wraps command ****N4ITKBiasFieldCorrection****

title: N4ITK MRI Bias correction

category: Filtering

description: Performs image bias correction using N4 algorithm. This module is based on the ITK filters contributed in the following publication: Tustison N, Gee J “N4ITK: Nick’s N3 ITK Implementation For MRI Bias Field Correction”, The Insight Journal 2009 January-June, <http://hdl.handle.net/10380/3053>

version: 9

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/N4ITKBiasFieldCorrection>

contributor: Nick Tustison (UPenn), Andrey Fedorov (SPL, BWH), Ron Kikinis (SPL, BWH)

acknowledgements: The development of this module was partially supported by NIH grants R01 AA016748-01, R01 CA111288 and U01 CA151261 as well as by NA-MIC, NAC, NCIGT and the Slicer community.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bsplineorder: (an integer (int or long))
    Order of B-spline used in the approximation. Larger values will lead
    to longer execution times, may result in overfitting and poor
    result.
    flag: --bsplineorder %d
convergencythreshold: (a float)
    Stopping criterion for the iterative bias estimation. Larger values
    will lead to smaller execution time.
    flag: --convergencythreshold %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
histogramsharpening: (a list of items which are a float)
    A vector of up to three values. Non-zero values correspond to Bias
    Field Full Width at Half Maximum, Wiener filter noise, and Number of
    histogram bins.
    flag: --histogramsharpening %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputimage: (an existing file name)
    Input image where you observe signal inhomogeneity
    flag: --inputimage %s
iterations: (a list of items which are an integer (int or long))
    Maximum number of iterations at each level of resolution. Larger
    values will increase execution time, but may lead to better results.
    flag: --iterations %s
maskimage: (an existing file name)
    Binary mask that defines the structure of your interest. NOTE: This
    parameter is OPTIONAL. If the mask is not specified, the module will
    use internally Otsu thresholding to define this mask. Better
    processing results can often be obtained when a meaningful mask is
    defined.
    flag: --maskimage %s
meshresolution: (a list of items which are a float)
    Resolution of the initial bspline grid defined as a sequence of
    three numbers. The actual resolution will be defined by adding the
    bspline order (default is 3) to the resolution in each dimension
    specified here. For example, 1,1,1 will result in a 4x4x4 grid of
    control points. This parameter may need to be adjusted based on your
    input image. In the multi-resolution N4 framework, the resolution of
    the bspline grid at subsequent iterations will be doubled. The
    number of resolutions is implicitly defined by Number of iterations
    parameter (the size of this list is the number of resolutions)
    flag: --meshresolution %s
outputbiasfield: (a boolean or a file name)
    Recovered bias field (OPTIONAL)
    flag: --outputbiasfield %s
outputimage: (a boolean or a file name)
    Result of processing
    flag: --outputimage %s

```

```

shrinkfactor: (an integer (int or long))
    Defines how much the image should be upsampled before estimating the
    inhomogeneity field. Increase if you want to reduce the execution
    time. 1 corresponds to the original resolution. Larger values will
    significantly reduce the computation time.
    flag: --shrinkfactor %d
splinedistance: (a float)
    An alternative means to define the spline grid, by setting the
    distance between the control points. This parameter is used only if
    the grid resolution is not specified.
    flag: --splinedistance %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
weightimage: (an existing file name)
    Weight Image
    flag: --weightimage %s

```

Outputs:

```

outputbiasfield: (an existing file name)
    Recovered bias field (OPTIONAL)
outputimage: (an existing file name)
    Result of processing

```

interfaces.slicer.filtering.resamplescalarvectordwivolume**ResampleScalarVectorDWIVolume**[Link to code](#)Wraps command ****ResampleScalarVectorDWIVolume ****

title: Resample Scalar/Vector/DWI Volume

category: Filtering

description: This module implements image and vector-image resampling through the use of itk Transforms. It can also handle diffusion weighted MRI image resampling. “Resampling” is performed in space coordinates, not pixel/grid coordinates. It is quite important to ensure that image spacing is properly set on the images involved. The interpolator is required since the mapping from one space to the other will often require evaluation of the intensity of the image at non-grid positions.

Warning: To resample DWMR Images, use nrrd input and output files.

Warning: Do not use to resample Diffusion Tensor Images, tensors would not be reoriented

version: 0.1

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ResampleScalarVectorDWIVolume>

contributor: Francois Budin (UNC)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. Information on the National Centers for Biomedical Computing can be obtained from <http://nihroadmap.nih.gov/bioinformatics>

Inputs:

```

[Mandatory]

[Optional]
Inverse_ITK_Transformation: (a boolean)

```



```

        Inverse the transformation before applying it from output image to
        input image
        flag: --Inverse_ITK_Transformation
Reference: (an existing file name)
        Reference Volume (spacing,size,orientation,origin)
        flag: --Reference %s
args: (a unicode string)
        Additional parameters to the command
        flag: %s
centered_transform: (a boolean)
        Set the center of the transformation to the center of the input
        image
        flag: --centered_transform
defField: (an existing file name)
        File containing the deformation field (3D vector image containing
        vectors with 3 components)
        flag: --defField %s
default_pixel_value: (a float)
        Default pixel value for samples falling outside of the input region
        flag: --default_pixel_value %f
direction_matrix: (a list of items which are a float)
        9 parameters of the direction matrix by rows (ijk to LPS if LPS
        transform, ijk to RAS if RAS transform)
        flag: --direction_matrix %s
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
hfieldtype: ('displacement' or 'h-Field')
        Set if the deformation field is an h-Field
        flag: --hfieldtype %s
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
image_center: ('input' or 'output')
        Image to use to center the transform (used only if 'Centered
        Transform' is selected)
        flag: --image_center %s
inputVolume: (an existing file name)
        Input Volume to be resampled
        flag: %s, position: -2
interpolation: ('linear' or 'nn' or 'ws' or 'bs')
        Sampling algorithm (linear or nn (nearest neighborhood), ws
        (WindowedSinc), bs (BSpline) )
        flag: --interpolation %s
notbulk: (a boolean)
        The transform following the BSpline transform is not set as a bulk
        transform for the BSpline transform
        flag: --notbulk
number_of_thread: (an integer (int or long))
        Number of thread used to compute the output image
        flag: --number_of_thread %d
origin: (a list of items which are any value)
        Origin of the output Image
        flag: --origin %s
outputVolume: (a boolean or a file name)
        Resampled Volume
        flag: %s, position: -1

```

```

rotation_point: (a list of items which are any value)
    Rotation Point in case of rotation around a point (otherwise
    useless)
    flag: --rotation_point %s
size: (a list of items which are a float)
    Size along each dimension (0 means use input size)
    flag: --size %s
spaceChange: (a boolean)
    Space Orientation between transform and image is different (RAS/LPS)
    (warning: if the transform is a Transform Node in Slicer3, do not
    select)
    flag: --spaceChange
spacing: (a list of items which are a float)
    Spacing along each dimension (0 means use input spacing)
    flag: --spacing %s
spline_order: (an integer (int or long))
    Spline Order
    flag: --spline_order %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transform: ('rt' or 'a')
    Transform algorithm, rt = Rigid Transform, a = Affine Transform
    flag: --transform %s
transform_matrix: (a list of items which are a float)
    12 parameters of the transform matrix by rows ( --last 3 being
    translation-- )
    flag: --transform_matrix %s
transform_order: ('input-to-output' or 'output-to-input')
    Select in what order the transforms are read
    flag: --transform_order %s
transformationFile: (an existing file name)
    flag: --transformationFile %s
window_function: ('h' or 'c' or 'w' or 'l' or 'b')
    Window Function , h = Hamming , c = Cosine , w = Welch , l = Lanczos
    , b = Blackman
    flag: --window_function %s

```

Outputs:

```

outputVolume: (an existing file name)
    Resampled Volume

```

interfaces.slicer.filtering.thresholdscalarvolume**ThresholdScalarVolume**[Link to code](#)Wraps command ****ThresholdScalarVolume ****

title: Threshold Scalar Volume

category: Filtering

description: **<p>Threshold an image.</p><p>Set image values to a user-specified outside value if they are below, above, or between simple threshold values.</p><p>ThresholdAbove: The values greater than or equal to the threshold value are set to OutsideValue.</p><p>ThresholdBelow: The values less than or equal to the threshold value are set to OutsideValue.</p><p>ThresholdOutside: The values outside the range Lower-Upper**

are set to OutsideValue.</p><p>Although all image types are supported on input, only signed types are produced.</p><p>

version: 0.1.0.\$Revision: 2104 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/Threshold>

contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
InputVolume: (an existing file name)
    Input volume
    flag: %s, position: -2
OutputVolume: (a boolean or a file name)
    Thresholded input volume
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
lower: (an integer (int or long))
    Lower threshold value
    flag: --lower %d
outsidevalue: (an integer (int or long))
    Set the voxels to this value if they fall outside the threshold
    range
    flag: --outsidevalue %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (an integer (int or long))
    Threshold value
    flag: --threshold %d
thresholdtype: ('Below' or 'Above' or 'Outside')
    What kind of threshold to perform. If Outside is selected, uses
    Upper and Lower values. If Below is selected, uses the
    ThresholdValue, if Above is selected, uses the ThresholdValue.
    flag: --thresholdtype %s
upper: (an integer (int or long))
    Upper threshold value
    flag: --upper %d
```

Outputs:

```
OutputVolume: (an existing file name)
    Thresholded input volume
```

interfaces.slicer.filtering.votingbinaryholefillingimagefilter

VotingBinaryHoleFillingImageFilter

[Link to code](#)

Wraps command ****VotingBinaryHoleFillingImageFilter****

title: Voting Binary Hole Filling Image Filter

category: Filtering

description: Applies a voting operation in order to fill-in cavities. This can be used for smoothing contours and for filling holes in binary images. This technique is used frequently when segmenting complete organs that may have ducts or vasculature that may not have been included in the initial segmentation, e.g. lungs, kidneys, liver.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/VotingBinaryHoleFillingImageFilter>

contributor: Bill Lorensen (GE)

acknowledgements: This command module was derived from Insight/Examples/Filtering/VotingBinaryHoleFillingImageFilter (copyright) Insight Software Consortium

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
background: (an integer (int or long))
    The value associated with the background (not object)
    flag: --background %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
foreground: (an integer (int or long))
    The value associated with the foreground (object)
    flag: --foreground %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume to be filtered
    flag: %s, position: -2
majorityThreshold: (an integer (int or long))
    The number of pixels over 50% that will decide whether an OFF pixel
    will become ON or not. For example, if the neighborhood of a pixel
    has 124 pixels (excluding itself), the 50% will be 62, and if you
    set a Majority threshold of 5, that means that the filter will
    require 67 or more neighbor pixels to be ON in order to switch the
    current OFF pixel to ON.
    flag: --majorityThreshold %d
outputVolume: (a boolean or a file name)
    Output filtered
    flag: %s, position: -1
radius: (a list of items which are an integer (int or long))
    The radius of a hole to be filled
    flag: --radius %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately

```

```
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputVolume: (an existing file name)
Output filtered
```

interfaces.slicer.legacy.converters

BSplineToDeformationField

[Link to code](#)

Wraps command ****BSplineToDeformationField****

title: BSpline to deformation field

category: Legacy.Converters

description: Create a dense deformation field from a bspline+bulk transform.

version: 0.1.0.\$Revision: 2104 \$(alpha)

documentation-url: [http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/](http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BSplineToDeformationField)

[BSplineToDeformationField](#)

contributor: Andrey Fedorov (SPL, BWH)

acknowledgements: This work is funded by NIH grants R01 CA111288 and U01 CA151261.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
defImage: (a boolean or a file name)
    flag: --defImage %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
refImage: (an existing file name)
    flag: --refImage %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
tfm: (an existing file name)
    flag: --tfm %s
```

Outputs:

```
defImage: (an existing file name)
```

interfaces.slicer.legacy.diffusion.denoising

DWIUnbiasedNonLocalMeansFilter

[Link to code](#)

Wraps command ****DWIUnbiasedNonLocalMeansFilter****

title: DWI Unbiased Non Local Means Filter

category: Legacy.Diffusion.Denoising

description: This module reduces noise (or unwanted detail) on a set of diffusion weighted images. For this, it filters the images using a Unbiased Non Local Means for Rician noise algorithm. It exploits not only the spatial redundancy, but the redundancy in similar gradient directions as well; it takes into account the N closest gradient directions to the direction being processed (a maximum of 5 gradient directions is allowed to keep a reasonable computational load, since we do not use neither similarity maps nor block-wise implementation). The noise parameter is automatically estimated in the same way as in the jointLMMSE module. A complete description of the algorithm may be found in: Antonio Tristan-Vega and Santiago Aja-Fernandez, DWI filtering using joint information for DTI and HARDI, Medical Image Analysis, Volume 14, Issue 2, Pages 205-218. 2010. Please, note that the execution of this filter is extremely slow, son only very conservative parameters (block size and search size as small as possible) should be used. Even so, its execution may take several hours. The advantage of this filter over joint LMMSE is its better preservation of edges and fine structures.

version: 0.0.1.\$Revision: 1 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/UnbiasedNonLocalMeansFilterForDWI>

contributor: Antonio Tristan Vega (UVa), Santiago Aja Fernandez (UVa)

acknowledgements: Partially founded by grant number TEC2007-67073/TCM from the Comision Interministerial de Ciencia y Tecnologia (Spain).

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
hp: (a float)
    This parameter is related to noise; the larger the parameter, the
    more aggressive the filtering. Should be near 1, and only values
    between 0.8 and 1.2 are allowed
    flag: --hp %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input DWI volume.
    flag: %s, position: -2
ng: (an integer (int or long))
    The number of the closest gradients that are used to jointly filter
    a given gradient direction (a maximum of 5 is allowed).
    flag: --ng %d
outputVolume: (a boolean or a file name)
    Output DWI volume.
    flag: %s, position: -1
rc: (a list of items which are an integer (int or long))
```

```

        Similarity between blocks is measured using windows of this size.
        flag: --rc %s
re: (a list of items which are an integer (int or long))
    A neighborhood of this size is used to compute the statistics for
    noise estimation.
    flag: --re %s
rs: (a list of items which are an integer (int or long))
    The algorithm search for similar voxels in a neighborhood of this
    size (larger sizes than the default one are extremely slow).
    flag: --rs %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Output DWI volume.

```

interfaces.slicer.legacy.filtering

OtsuThresholdImageFilter

[Link to code](#)Wraps command ****OtsuThresholdImageFilter ****

title: Otsu Threshold Image Filter

category: Legacy.Filtering

description: This filter creates a binary thresholded image that separates an image into foreground and background components. The filter calculates the optimum threshold separating those two classes so that their combined spread (intra-class variance) is minimal (see http://en.wikipedia.org/wiki/Otsu%27s_method). Then the filter applies that threshold to the input image using the itkBinaryThresholdImageFilter. The numberOfHistogram bins can be set for the Otsu Calculator. The insideValue and outsideValue can be set for the BinaryThresholdImageFilter. The filter produces a labeled volume.

The original reference is:

N.Otsu, A threshold selection method from gray level histograms, IEEE Trans.Syst.ManCybern.SMC-9,62–66 1979.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: [http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/](http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/OtsuThresholdImageFilter)[OtsuThresholdImageFilter](#)

contributor: Bill Lorensen (GE)

acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software Consortium

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables

```

```

ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume to be filtered
    flag: %s, position: -2
insideValue: (an integer (int or long))
    The value assigned to pixels that are inside the computed threshold
    flag: --insideValue %d
numberOfBins: (an integer (int or long))
    This is an advanced parameter. The number of bins in the histogram
    used to model the probability mass function of the two intensity
    distributions. Small numbers of bins may result in a more
    conservative threshold. The default should suffice for most
    applications. Experimentation is the only way to see the effect of
    varying this parameter.
    flag: --numberOfBins %d
outputVolume: (a boolean or a file name)
    Output filtered
    flag: %s, position: -1
outsideValue: (an integer (int or long))
    The value assigned to pixels that are outside the computed threshold
    flag: --outsideValue %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Output filtered

```

ResampleScalarVolume[Link to code](#)Wraps command ****ResampleScalarVolume ****

title: Resample Scalar Volume

category: Legacy.Filtering

description: Resampling an image is an important task in image analysis. It is especially important in the frame of image registration. This module implements image resampling through the use of itk Transforms. This module uses an Identity Transform. The resampling is controlled by the Output Spacing. “Resampling” is performed in space coordinates, not pixel/grid coordinates. It is quite important to ensure that image spacing is properly set on the images involved. The interpolator is required since the mapping from one space to the other will often require evaluation of the intensity of the image at non-grid positions. Several interpolators are available: linear, nearest neighbor, bspline and five flavors of sinc. The sinc interpolators, although more precise, are much slower than the linear and nearest neighbor interpolator. To resample label volumnes, nearest neighbor interpolation should be used exclusively.

version: 0.1.0.\$Revision: 20594 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ResampleVolume>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:


```

[Mandatory]

[Optional]
InputVolume: (an existing file name)
    Input volume to be resampled
    flag: %s, position: -2
OutputVolume: (a boolean or a file name)
    Resampled Volume
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interpolation: ('linear' or 'nearestNeighbor' or 'bspline' or
    'hamming' or 'cosine' or 'welch' or 'lanczos' or 'blackman')
    Sampling algorithm (linear, nearest neighbor, bspline(cubic) or
    windowed sinc). There are several sinc algorithms available as
    described in the following publication: Erik H. W. Meijering, Wiro
    J. Niessen, Josien P. W. Pluim, Max A. Viergever: Quantitative
    Comparison of Sinc-Approximating Kernels for Medical Image
    Interpolation. MICCAI 1999, pp. 210-217. Each window has a radius of
    3;
    flag: --interpolation %s
spacing: (a list of items which are a float)
    Spacing along each dimension (0 means use input spacing)
    flag: --spacing %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

OutputVolume: (an existing file name)
    Resampled Volume

```

interfaces.slicer.legacy.registration

AffineRegistration

[Link to code](#)Wraps command ****AffineRegistration****

title: Affine Registration

category: Legacy.Registration

description: Registers two images together using an affine transform and mutual information. This module is often used to align images of different subjects or images of the same subject from different modalities.

This module can smooth images prior to registration to mitigate noise and improve convergence. Many of the registration parameters require a working knowledge of the algorithm although the default parameters are sufficient for many registration tasks.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/AffineRegistration>

contributor: Daniel Blezek (GE)

acknowledgements: This module was developed by Daniel Blezek while at GE Research with contributions from Jim Miller.

This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
FixedImageFileName: (an existing file name)
    Fixed image to which to register
    flag: %s, position: -2
MovingImageFileName: (an existing file name)
    Moving image
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixedsmoothingfactor: (an integer (int or long))
    Amount of smoothing applied to fixed image prior to registration.
    Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the
    input data if there is considerable amounts of noise or the noise
    pattern in the fixed and moving images is very different.
    flag: --fixedsmoothingfactor %d
histogrambins: (an integer (int or long))
    Number of histogram bins to use for Mattes Mutual Information.
    Reduce the number of bins if a registration fails. If the number of
    bins is too large, the estimated PDFs will be a field of impulses
    and will inhibit reliable registration estimation.
    flag: --histogrambins %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initialtransform: (an existing file name)
    Initial transform for aligning the fixed and moving image. Maps
    positions in the fixed coordinate frame to positions in the moving
    coordinate frame. Optional.
    flag: --initialtransform %s
iterations: (an integer (int or long))
    Number of iterations
    flag: --iterations %d
movingsmoothingfactor: (an integer (int or long))
    Amount of smoothing applied to moving image prior to registration.
    Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the
    input data if there is considerable amounts of noise or the noise
    pattern in the fixed and moving images is very different.
    flag: --movingsmoothingfactor %d
outputtransform: (a boolean or a file name)
    Transform calculated that aligns the fixed and moving image. Maps
    positions in the fixed coordinate frame to the moving coordinate
    frame. Optional (specify an output transform or an output volume or
    both).
```

```

    flag: --outputtransform %s
resampledmovingfilename: (a boolean or a file name)
    Resampled moving image to the fixed image coordinate frame. Optional
    (specify an output transform or an output volume or both).
    flag: --resampledmovingfilename %s
spatialsamples: (an integer (int or long))
    Number of spatial samples to use in estimating Mattes Mutual
    Information. Larger values yield more accurate PDFs and improved
    registration quality.
    flag: --spatialsamples %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
translationscale: (a float)
    Relative scale of translations to rotations, i.e. a value of 100
    means 10mm = 1 degree. (Actual scale used is
    1/(TranslationScale^2)). This parameter is used to 'weight' or
    'standardized' the transform parameters and their effect on the
    registration objective function.
    flag: --translationscale %f

```

Outputs:

```

outputtransform: (an existing file name)
    Transform calculated that aligns the fixed and moving image. Maps
    positions in the fixed coordinate frame to the moving coordinate
    frame. Optional (specify an output transform or an output volume or
    both).
resampledmovingfilename: (an existing file name)
    Resampled moving image to the fixed image coordinate frame. Optional
    (specify an output transform or an output volume or both).

```

BSplineDeformableRegistration[Link to code](#)Wraps command ****BSplineDeformableRegistration****

title: BSpline Deformable Registration

category: Legacy.Registration

description: Registers two images together using BSpline transform and mutual information.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/BSplineDeformableRegistration>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
FixedImageFileName: (an existing file name)
    Fixed image to which to register
    flag: %s, position: -2
MovingImageFileName: (an existing file name)
    Moving image
    flag: %s, position: -1

```

```
args: (a unicode string)
    Additional parameters to the command
    flag: %s
constrain: (a boolean)
    Constrain the deformation to the amount specified in Maximum
    Deformation
    flag: --constrain
default: (an integer (int or long))
    Default pixel value used if resampling a pixel outside of the
    volume.
    flag: --default %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gridSize: (an integer (int or long))
    Number of grid points on interior of the fixed image. Larger grid
    sizes allow for finer registrations.
    flag: --gridSize %d
histogrambins: (an integer (int or long))
    Number of histogram bins to use for Mattes Mutual Information.
    Reduce the number of bins if a deformable registration fails. If the
    number of bins is too large, the estimated PDFs will be a field of
    impulses and will inhibit reliable registration estimation.
    flag: --histogrambins %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initialtransform: (an existing file name)
    Initial transform for aligning the fixed and moving image. Maps
    positions in the fixed coordinate frame to positions in the moving
    coordinate frame. This transform should be an affine or rigid
    transform. It is used as a bulk transform for the BSpline. Optional.
    flag: --initialtransform %s
iterations: (an integer (int or long))
    Number of iterations
    flag: --iterations %d
maximumDeformation: (a float)
    If Constrain Deformation is checked, limit the deformation to this
    amount.
    flag: --maximumDeformation %f
outputtransform: (a boolean or a file name)
    Transform calculated that aligns the fixed and moving image. Maps
    positions from the fixed coordinate frame to the moving coordinate
    frame. Optional (specify an output transform or an output volume or
    both).
    flag: --outputtransform %s
outputwarp: (a boolean or a file name)
    Vector field that applies an equivalent warp as the BSpline. Maps
    positions from the fixed coordinate frame to the moving coordinate
    frame. Optional.
    flag: --outputwarp %s
resampledmovingfilename: (a boolean or a file name)
    Resampled moving image to fixed image coordinate frame. Optional
    (specify an output transform or an output volume or both).
    flag: --resampledmovingfilename %s
spatialsamples: (an integer (int or long))
    Number of spatial samples to use in estimating Mattes Mutual
```

```

Information. Larger values yield more accurate PDFs and improved
registration quality.
flag: --spatialsamples %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputtransform: (an existing file name)
    Transform calculated that aligns the fixed and moving image. Maps
    positions from the fixed coordinate frame to the moving coordinate
    frame. Optional (specify an output transform or an output volume or
    both).
outputwarp: (an existing file name)
    Vector field that applies an equivalent warp as the BSpline. Maps
    positions from the fixed coordinate frame to the moving coordinate
    frame. Optional.
resampledmovingfilename: (an existing file name)
    Resampled moving image to fixed image coordinate frame. Optional
    (specify an output transform or an output volume or both).

```

ExpertAutomatedRegistration

Link to code

Wraps command ****ExpertAutomatedRegistration ****

title: Expert Automated Registration

category: Legacy.Registration

description: Provides rigid, affine, and BSpline registration methods via a simple GUI

version: 0.1.0.\$Revision: 2104 \$(alpha)

documentation-url: [http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/](http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ExpertAutomatedRegistration)

ExpertAutomatedRegistration

contributor: Stephen R Aylward (Kitware), Casey B Goodlett (Kitware)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
affineMaxIterations: (an integer (int or long))
    Maximum number of affine optimization iterations
    flag: --affineMaxIterations %d
affineSamplingRatio: (a float)
    Portion of the image to use in computing the metric during affine
    registration
    flag: --affineSamplingRatio %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bsplineMaxIterations: (an integer (int or long))
    Maximum number of bspline optimization iterations
    flag: --bsplineMaxIterations %d
bsplineSamplingRatio: (a float)
    Portion of the image to use in computing the metric during BSpline
    registration

```

```

    flag: --bsplineSamplingRatio %f
controlPointSpacing: (an integer (int or long))
    Number of pixels between control points
    flag: --controlPointSpacing %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
expectedOffset: (a float)
    Expected misalignment after initialization
    flag: --expectedOffset %f
expectedRotation: (a float)
    Expected misalignment after initialization
    flag: --expectedRotation %f
expectedScale: (a float)
    Expected misalignment after initialization
    flag: --expectedScale %f
expectedSkew: (a float)
    Expected misalignment after initialization
    flag: --expectedSkew %f
fixedImage: (an existing file name)
    Image which defines the space into which the moving image is
    registered
    flag: %s, position: -2
fixedImageMask: (an existing file name)
    Image which defines a mask for the fixed image
    flag: --fixedImageMask %s
fixedLandmarks: (a list of items which are a list of from 3 to 3
    items which are a float)
    Ordered list of landmarks in the fixed image
    flag: --fixedLandmarks %s...
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initialization: ('None' or 'Landmarks' or 'ImageCenters' or
    'CentersOfMass' or 'SecondMoments')
    Method to prime the registration process
    flag: --initialization %s
interpolation: ('NearestNeighbor' or 'Linear' or 'BSpline')
    Method for interpolation within the optimization process
    flag: --interpolation %s
loadTransform: (an existing file name)
    Load a transform that is immediately applied to the moving image
    flag: --loadTransform %s
metric: ('MattesMI' or 'NormCorr' or 'MeanSqr  ')
    Method to quantify image match
    flag: --metric %s
minimizeMemory: (a boolean)
    Reduce the amount of memory required at the cost of increased
    computation time
    flag: --minimizeMemory
movingImage: (an existing file name)
    The transform goes from the fixed image's space into the moving
    image's space
    flag: %s, position: -1
movingLandmarks: (a list of items which are a list of from 3 to 3
    items which are a float)
    Ordered list of landmarks in the moving image

```

```

    flag: --movingLandmarks %s...
numberOfThreads: (an integer (int or long))
    Number of CPU threads to use
    flag: --numberOfThreads %d
randomNumberSeed: (an integer (int or long))
    Seed to generate a consistent random number sequence
    flag: --randomNumberSeed %d
registration: ('None' or 'Initial' or 'Rigid' or 'Affine' or
    'BSpline' or 'PipelineRigid' or 'PipelineAffine' or
    'PipelineBSpline')
    Method for the registration process
    flag: --registration %s
resampledImage: (a boolean or a file name)
    Registration results
    flag: --resampledImage %s
rigidMaxIterations: (an integer (int or long))
    Maximum number of rigid optimization iterations
    flag: --rigidMaxIterations %d
rigidSamplingRatio: (a float)
    Portion of the image to use in computing the metric during rigid
    registration
    flag: --rigidSamplingRatio %f
sampleFromOverlap: (a boolean)
    Limit metric evaluation to the fixed image region overlapped by the
    moving image
    flag: --sampleFromOverlap
saveTransform: (a boolean or a file name)
    Save the transform that results from registration
    flag: --saveTransform %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbosityLevel: ('Silent' or 'Standard' or 'Verbose')
    Level of detail of reporting progress
    flag: --verbosityLevel %s

```

Outputs:

```

resampledImage: (an existing file name)
    Registration results
saveTransform: (an existing file name)
    Save the transform that results from registration

```

LinearRegistration[Link to code](#)Wraps command ****LinearRegistration****

title: Linear Registration

category: Legacy.Registration

description: Registers two images together using a rigid transform and mutual information.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/LinearRegistration>

contributor: Daniel Blezek (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
FixedImageFileName: (an existing file name)
    Fixed image to which to register
    flag: %s, position: -2
MovingImageFileName: (an existing file name)
    Moving image
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixedsmoothingfactor: (an integer (int or long))
    Amount of smoothing applied to fixed image prior to registration.
    Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the
    input data if there is considerable amounts of noise or the noise
    pattern in the fixed and moving images is very different.
    flag: --fixedsmoothingfactor %d
histogrambins: (an integer (int or long))
    Number of histogram bins to use for Mattes Mutual Information.
    Reduce the number of bins if a registration fails. If the number of
    bins is too large, the estimated PDFs will be a field of impulses
    and will inhibit reliable registration estimation.
    flag: --histogrambins %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initialtransform: (an existing file name)
    Initial transform for aligning the fixed and moving image. Maps
    positions in the fixed coordinate frame to positions in the moving
    coordinate frame. Optional.
    flag: --initialtransform %s
iterations: (a list of items which are an integer (int or long))
    Comma separated list of iterations. Must have the same number of
    elements as the learning rate.
    flag: --iterations %s
learningrate: (a list of items which are a float)
    Comma separated list of learning rates. Learning rate is a scale
    factor on the gradient of the registration objective function
    (gradient with respect to the parameters of the transformation) used
    to update the parameters of the transformation during optimization.
    Smaller values cause the optimizer to take smaller steps through the
    parameter space. Larger values are typically used early in the
    registration process to take large jumps in parameter space followed
    by smaller values to home in on the optimum value of the
    registration objective function. Default is: 0.01, 0.005, 0.0005,
    0.0002. Must have the same number of elements as iterations.
    flag: --learningrate %s
movingsmoothingfactor: (an integer (int or long))
    Amount of smoothing applied to moving image prior to registration.
    Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the
    input data if there is considerable amounts of noise or the noise
    pattern in the fixed and moving images is very different.
    flag: --movingsmoothingfactor %d

```



```

outputtransform: (a boolean or a file name)
    Transform calculated that aligns the fixed and moving image. Maps
    positions in the fixed coordinate frame to the moving coordinate
    frame. Optional (specify an output transform or an output volume or
    both).
    flag: --outputtransform %s
resampledmovingfilename: (a boolean or a file name)
    Resampled moving image to the fixed image coordinate frame. Optional
    (specify an output transform or an output volume or both).
    flag: --resampledmovingfilename %s
spatialsamples: (an integer (int or long))
    Number of spatial samples to use in estimating Mattes Mutual
    Information. Larger values yield more accurate PDFs and improved
    registration quality.
    flag: --spatialsamples %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
translationscale: (a float)
    Relative scale of translations to rotations, i.e. a value of 100
    means 10mm = 1 degree. (Actual scale used 1/(TranslationScale^2)).
    This parameter is used to 'weight' or 'standardized' the transform
    parameters and their effect on the registration objective function.
    flag: --translationscale %f

```

Outputs:

```

outputtransform: (an existing file name)
    Transform calculated that aligns the fixed and moving image. Maps
    positions in the fixed coordinate frame to the moving coordinate
    frame. Optional (specify an output transform or an output volume or
    both).
resampledmovingfilename: (an existing file name)
    Resampled moving image to the fixed image coordinate frame. Optional
    (specify an output transform or an output volume or both).

```

MultiResolutionAffineRegistration

[Link to code](#)

Wraps command ****MultiResolutionAffineRegistration****

title: Robust Multiresolution Affine Registration

category: Legacy.Registration

description: Provides affine registration using multiple resolution levels and decomposed affine transforms.

version: 0.1.0.\$Revision: 2104 \$(alpha)

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/MultiResolutionAffineRegistration>

contributor: Casey B Goodlett (Utah)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command

```

```

    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
fixedImage: (an existing file name)
    Image which defines the space into which the moving image is
    registered
    flag: %s, position: -2
fixedImageMask: (an existing file name)
    Label image which defines a mask of interest for the fixed image
    flag: --fixedImageMask %s
fixedImageROI: (a list of items which are any value)
    Label image which defines a ROI of interest for the fixed image
    flag: --fixedImageROI %s
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
metricTolerance: (a float)
    flag: --metricTolerance %f
movingImage: (an existing file name)
    The transform goes from the fixed image's space into the moving
    image's space
    flag: %s, position: -1
numIterations: (an integer (int or long))
    Number of iterations to run at each resolution level.
    flag: --numIterations %d
numLineIterations: (an integer (int or long))
    Number of iterations to run at each resolution level.
    flag: --numLineIterations %d
resampledImage: (a boolean or a file name)
    Registration results
    flag: --resampledImage %s
saveTransform: (a boolean or a file name)
    Save the output transform from the registration
    flag: --saveTransform %s
stepSize: (a float)
    The maximum step size of the optimizer in voxels
    flag: --stepSize %f
stepTolerance: (a float)
    The maximum step size of the optimizer in voxels
    flag: --stepTolerance %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

resampledImage: (an existing file name)
    Registration results
saveTransform: (an existing file name)
    Save the output transform from the registration

```

RigidRegistration[Link to code](#)Wraps command ****RigidRegistration****

title: Rigid Registration

category: Legacy.Registration

description: Registers two images together using a rigid transform and mutual information.

This module was originally distributed as “Linear registration” but has been renamed to eliminate confusion with the “Affine registration” module.

This module is often used to align images of different subjects or images of the same subject from different modalities.

This module can smooth images prior to registration to mitigate noise and improve convergence. Many of the registration parameters require a working knowledge of the algorithm although the default parameters are sufficient for many registration tasks.

version: 0.1.0.\$Revision: 19608 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/RigidRegistration>

contributor: Daniel Blezek (GE)

acknowledgements: This module was developed by Daniel Blezek while at GE Research with contributions from Jim Miller.

This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
FixedImageFileName: (an existing file name)
    Fixed image to which to register
    flag: %s, position: -2
MovingImageFileName: (an existing file name)
    Moving image
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixedsmoothingfactor: (an integer (int or long))
    Amount of smoothing applied to fixed image prior to registration.
    Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the
    input data if there is considerable amounts of noise or the noise
    pattern in the fixed and moving images is very different.
    flag: --fixedsmoothingfactor %d
histogrambins: (an integer (int or long))
    Number of histogram bins to use for Mattes Mutual Information.
    Reduce the number of bins if a registration fails. If the number of
    bins is too large, the estimated PDFs will be a field of impulses
    and will inhibit reliable registration estimation.
    flag: --histogrambins %d
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initialtransform: (an existing file name)
    Initial transform for aligning the fixed and moving image. Maps
    positions in the fixed coordinate frame to positions in the moving
    coordinate frame. Optional.
    flag: --initialtransform %s
iterations: (a list of items which are an integer (int or long))
    Comma separated list of iterations. Must have the same number of
```

```

elements as the learning rate.
flag: --iterations %s
learningrate: (a list of items which are a float)
    Comma separated list of learning rates. Learning rate is a scale
    factor on the gradient of the registration objective function
    (gradient with respect to the parameters of the transformation) used
    to update the parameters of the transformation during optimization.
    Smaller values cause the optimizer to take smaller steps through the
    parameter space. Larger values are typically used early in the
    registration process to take large jumps in parameter space followed
    by smaller values to home in on the optimum value of the
    registration objective function. Default is: 0.01, 0.005, 0.0005,
    0.0002. Must have the same number of elements as iterations.
    flag: --learningrate %s
movingsmoothingfactor: (an integer (int or long))
    Amount of smoothing applied to moving image prior to registration.
    Default is 0 (none). Range is 0-5 (unitless). Consider smoothing the
    input data if there is considerable amounts of noise or the noise
    pattern in the fixed and moving images is very different.
    flag: --movingsmoothingfactor %d
outputtransform: (a boolean or a file name)
    Transform calculated that aligns the fixed and moving image. Maps
    positions in the fixed coordinate frame to the moving coordinate
    frame. Optional (specify an output transform or an output volume or
    both).
    flag: --outputtransform %s
resampledmovingfilename: (a boolean or a file name)
    Resampled moving image to the fixed image coordinate frame. Optional
    (specify an output transform or an output volume or both).
    flag: --resampledmovingfilename %s
spatialsamples: (an integer (int or long))
    Number of spatial samples to use in estimating Mattes Mutual
    Information. Larger values yield more accurate PDFs and improved
    registration quality.
    flag: --spatialsamples %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
testingmode: (a boolean)
    Enable testing mode. Input transform will be used to construct
    floating image. The floating image will be ignored if passed.
    flag: --testingmode
translationscale: (a float)
    Relative scale of translations to rotations, i.e. a value of 100
    means 10mm = 1 degree. (Actual scale used 1/(TranslationScale^2)).
    This parameter is used to 'weight' or 'standardized' the transform
    parameters and their effect on the registration objective function.
    flag: --translationscale %f

```

Outputs:

```

outputtransform: (an existing file name)
    Transform calculated that aligns the fixed and moving image. Maps
    positions in the fixed coordinate frame to the moving coordinate
    frame. Optional (specify an output transform or an output volume or
    both).
resampledmovingfilename: (an existing file name)

```

Resampled moving image to the fixed image coordinate frame. Optional (specify an output transform **or** an output volume **or** both).

interfaces.slicer.legacy.segmentation

OtsuThresholdSegmentation

[Link to code](#)

Wraps command ****OtsuThresholdSegmentation****

title: Otsu Threshold Segmentation

category: Legacy.Segmentation

description: This filter creates a labeled image from a grayscale image. First, it calculates an optimal threshold that separates the image into foreground and background. This threshold separates those two classes so that their intra-class variance is minimal (see http://en.wikipedia.org/wiki/Otsu%27s_method). Then the filter runs a connected component algorithm to generate unique labels for each connected region of the foreground. Finally, the resulting image is relabeled to provide consecutive numbering.

version: 1.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/OtsuThresholdSegmentation>

contributor: Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
brightObjects: (a boolean)
    Segmenting bright objects on a dark background or dark objects on a
    bright background.
    flag: --brightObjects
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
faceConnected: (a boolean)
    This is an advanced parameter. Adjacent voxels are face connected.
    This affects the connected component algorithm. If this parameter is
    false, more regions are likely to be identified.
    flag: --faceConnected
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume to be segmented
    flag: %s, position: -2
minimumObjectSize: (an integer (int or long))
    Minimum size of object to retain. This parameter can be used to get
    rid of small regions in noisy images.
    flag: --minimumObjectSize %d
numberOfBins: (an integer (int or long))
    This is an advanced parameter. The number of bins in the histogram
```

```

        used to model the probability mass function of the two intensity
        distributions. Small numbers of bins may result in a more
        conservative threshold. The default should suffice for most
        applications. Experimentation is the only way to see the effect of
        varying this parameter.
        flag: --numberOfBins %d
outputVolume: (a boolean or a file name)
        Output filtered
        flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
        Control terminal output: `stream` - displays to terminal immediately
        (default), `allatonce` - waits till command is finished to display
        output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
        Output filtered

```

interfaces.slicer.quantification.changequantification**IntensityDifferenceMetric**[Link to code](#)Wraps command ****IntensityDifferenceMetric******title:** Intensity Difference Change Detection (FAST)**category:** Quantification.ChangeQuantification**description:** Quantifies the changes between two spatially aligned images based on the pixel-wise difference of image intensities.

version: 0.1

contributor: Andrey Fedorov

acknowledgements:

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
        Additional parameters to the command
        flag: %s
baselineSegmentationVolume: (an existing file name)
        Label volume that contains segmentation of the structure of interest
        in the baseline volume.
        flag: %s, position: -3
baselineVolume: (an existing file name)
        Baseline volume to be compared to
        flag: %s, position: -4
changingBandSize: (an integer (int or long))
        How far (in mm) from the boundary of the segmentation should the
        intensity changes be considered.
        flag: --changingBandSize %d
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a value
        of class 'str', nipy default value: {})
        Environment variables
followupVolume: (an existing file name)

```

```

    Followup volume to be compare to the baseline
    flag: %s, position: -2
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
outputVolume: (a boolean or a file name)
    Output volume to keep the results of change quantification.
    flag: %s, position: -1
reportFileName: (a boolean or a file name)
    Report file name
    flag: --reportFileName %s
sensitivityThreshold: (a float)
    This parameter should be between 0 and 1, and defines how sensitive
    the metric should be to the intensity changes.
    flag: --sensitivityThreshold %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Output volume to keep the results of change quantification.
reportFileName: (an existing file name)
    Report file name

```

interfaces.slicer.quantification.petstandarduptakevaluecomputation

PETStandardUptakeValueComputation

[Link to code](#)Wraps command ****PETStandardUptakeValueComputation ****

title: PET Standard Uptake Value Computation

category: Quantification

description: Computes the standardized uptake value based on body weight. Takes an input PET image in DICOM and NRRD format (DICOM header must contain Radiopharmaceutical parameters). Produces a CSV file that contains patientID, studyDate, dose, labelID, suvmin, suvmax, suvmean, labelName for each volume of interest. It also displays some of the information as output strings in the GUI, the CSV file is optional in that case. The CSV file is appended to on each execution of the CLI.

version: 0.1.0.\$Revision: 8595 \$(alpha)

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ComputeSUVBodyWeight>

contributor: Wendy Plesniak (SPL, BWH), Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)

acknowledgements: This work is funded by the Harvard Catalyst, and the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
OutputLabel: (a unicode string)
    List of labels for which SUV values were computed
    flag: --OutputLabel %s
OutputLabelValue: (a unicode string)

```

```

    List of label values for which SUV values were computed
    flag: --OutputLabelValue %s
SUVMax: (a unicode string)
    SUV max for each label
    flag: --SUVMax %s
SUVMean: (a unicode string)
    SUV mean for each label
    flag: --SUVMean %s
SUVMin: (a unicode string)
    SUV minimum for each label
    flag: --SUVMin %s
args: (a unicode string)
    Additional parameters to the command
    flag: %s
color: (an existing file name)
    Color table to to map labels to colors and names
    flag: --color %s
csvFile: (a boolean or a file name)
    A file holding the output SUV values in comma separated lines, one
    per label. Optional.
    flag: --csvFile %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
labelMap: (an existing file name)
    Input label volume containing the volumes of interest
    flag: --labelMap %s
petDICOMPath: (an existing directory name)
    Input path to a directory containing a PET volume containing DICOM
    header information for SUV computation
    flag: --petDICOMPath %s
petVolume: (an existing file name)
    Input PET volume for SUVbw computation (must be the same volume as
    pointed to by the DICOM path!).
    flag: --petVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

csvFile: (an existing file name)
    A file holding the output SUV values in comma separated lines, one
    per label. Optional.

```

interfaces.slicer.registration.brainsfit**BRAINSFit**[Link to code](#)Wraps command ****BRAINSFit** **

title: General Registration (BRAINS)

category: Registration

description: Register a three-dimensional volume to a reference volume (Mattes Mutual Information by default). Described in BRAINSFit: Mutual Information Registrations of Whole-Brain 3D Images, Using the Insight Toolkit, Johnson H.J., Harris G., Williams K., The Insight Journal, 2007. <http://hdl.handle.net/1926/1291>

version: 3.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:BRAINSFit>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: Hans J. Johnson, hans-johnson -at- uiowa.edu, <http://www.psychiatry.uiowa.edu>

acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); Gregory Harris(1), Vincent Magnotta(1,2,3); Andriy Fedorov(5) 1=University of Iowa Department of Psychiatry, 2=University of Iowa Department of Radiology, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering, 5=Surgical Planning Lab, Harvard

Inputs:

```
[Mandatory]

[Optional]
NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_00: (a boolean)
    DO NOT USE THIS FLAG
    flag: --NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_00
NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_01: (a boolean)
    DO NOT USE THIS FLAG
    flag: --NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_01
NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_02: (a boolean)
    DO NOT USE THIS FLAG
    flag: --NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_02
ROIAutoClosingSize: (a float)
    This flag is only relevant when using ROIAUTO mode for initializing
    masks. It defines the hole closing size in mm. It is rounded up to
    the nearest whole pixel size in each direction. The default is to
    use a closing size of 9mm. For mouse data this value may need to be
    reset to 0.9 or smaller.
    flag: --ROIAutoClosingSize %f
ROIAutoDilateSize: (a float)
    This flag is only relevant when using ROIAUTO mode for initializing
    masks. It defines the final dilation size to capture a bit of
    background outside the tissue region. At setting of 10mm has been
    shown to help regularize a BSpline registration type so that there
    is some background constraints to match the edges of the head
    better.
    flag: --ROIAutoDilateSize %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
backgroundFillValue: (a float)
    Background fill value for output image.
    flag: --backgroundFillValue %f
bsplineTransform: (a boolean or a file name)
    (optional) Filename to which save the estimated transform. NOTE: You
    must set at least one output object (either a deformed image or a
    transform. NOTE: USE THIS ONLY IF THE FINAL TRANSFORM IS BSpline
    flag: --bsplineTransform %s
costFunctionConvergenceFactor: (a float)
    From itkLBFGSBOptimizer.h: Set/Get the
    CostFunctionConvergenceFactor. Algorithm terminates when the
    reduction in cost function is less than (factor * epsmcj) where
    epsmcj is the machine precision. Typical values for factor: 1e+12
    for low accuracy; 1e+7 for moderate accuracy and 1e+1 for extremely
```

```

    high accuracy. 1e+9 seems to work well.,
    flag: --costFunctionConvergenceFactor %f
costMetric: ('MMI' or 'MSE' or 'NC' or 'MC')
    The cost metric to be used during fitting. Defaults to MMI. Options
    are MMI (Mattes Mutual Information), MSE (Mean Square Error), NC
    (Normalized Correlation), MC (Match Cardinality for binary images)
    flag: --costMetric %s
debugLevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
    flag: --debugLevel %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
failureExitCode: (an integer (int or long))
    If the fit fails, exit with this status code. (It can be used to
    force a successfull exit status of (0) if the registration fails due
    to reaching the maximum number of iterations.
    flag: --failureExitCode %d
fixedBinaryVolume: (an existing file name)
    Fixed Image binary mask volume, ONLY FOR MANUAL ROI mode.
    flag: --fixedBinaryVolume %s
fixedVolume: (an existing file name)
    The fixed image for registration by mutual information optimization.
    flag: --fixedVolume %s
fixedVolumeTimeIndex: (an integer (int or long))
    The index in the time series for the 3D fixed image to fit, if
    4-dimensional.
    flag: --fixedVolumeTimeIndex %d
forceMINumberOfThreads: (an integer (int or long))
    Force the the maximum number of threads to use for non thread safe
    MI metric. CAUTION: Inconsistent results my arise!
    flag: --forceMINumberOfThreads %d
gui: (a boolean)
    Display intermediate image volumes for debugging. NOTE: This is not
    part of the standard build sytem, and probably does nothing on your
    installation.
    flag: --gui
histogramMatch: (a boolean)
    Histogram Match the input images. This is suitable for images of the
    same modality that may have different absolute scales, but the same
    overall intensity profile. Do NOT use if registering images from
    different modailties.
    flag: --histogramMatch
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initialTransform: (an existing file name)
    Filename of transform used to initialize the registration. This CAN
    NOT be used with either CenterOfHeadLAlign, MomentsAlign,
    GeometryAlign, or initialTransform file.
    flag: --initialTransform %s
initializeTransformMode: ('Off' or 'useMomentsAlign' or
    'useCenterOfHeadAlign' or 'useGeometryAlign' or
    'useCenterOfROIAlign')
    Determine how to initialize the transform center. GeometryAlign on
    assumes that the center of the voxel lattice of the images represent

```

```

similar structures. MomentsAlign assumes that the center of mass of
the images represent similar structures. useCenterOfHeadAlign
attempts to use the top of head and shape of neck to drive a center
of mass estimate. Off assumes that the physical space of the images
are close, and that centering in terms of the image Origins is a
good starting point. This flag is mutually exclusive with the
initialTransform flag.
flag: --initializeTransformMode %s
interpolationMode: ('NearestNeighbor' or 'Linear' or
  'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
  'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
Type of interpolation to be used when applying transform to moving
volume. Options are Linear, NearestNeighbor, BSpline, WindowedSinc,
or ResampleInPlace. The ResampleInPlace option will create an image
with the same discrete voxel values and will adjust the origin and
direction of the physical space interpretation.
flag: --interpolationMode %s
linearTransform: (a boolean or a file name)
  (optional) Filename to which save the estimated transform. NOTE: You
  must set at least one output object (either a deformed image or a
  transform. NOTE: USE THIS ONLY IF THE FINAL TRANSFORM IS ---NOT---
  BSpline
  flag: --linearTransform %s
maskInferiorCutOffFromCenter: (a float)
  For use with --useCenterOfHeadAlign (and --maskProcessingMode
  ROIAUTO): the cut-off below the image centers, in millimeters,
  flag: --maskInferiorCutOffFromCenter %f
maskProcessingMode: ('NOMASK' or 'ROIAUTO' or 'ROI')
  What mode to use for using the masks. If ROIAUTO is choosen, then
  the mask is implicitly defined using a otsu foreground and hole
  filling algorithm. The Region Of Interest mode (choose ROI) uses the
  masks to define what parts of the image should be used for computing
  the transform.
  flag: --maskProcessingMode %s
maxBSplineDisplacement: (a float)
  Sets the maximum allowed displacements in image physical
  coordinates for BSpline control grid along each axis. A value of 0.0
  indicates that the problem should be unbounded. NOTE: This only
  constrains the BSpline portion, and does not limit the displacement
  from the associated bulk transform. This can lead to a substantial
  reduction in computation time in the BSpline optimizer.,
  flag: --maxBSplineDisplacement %f
maximumStepLength: (a float)
  Internal debugging parameter, and should probably never be used from
  the command line. This will be removed in the future.
  flag: --maximumStepLength %f
medianFilterSize: (a list of items which are an integer (int or
  long))
  The radius for the optional MedianImageFilter preprocessing in all 3
  directions.
  flag: --medianFilterSize %s
minimumStepLength: (a list of items which are a float)
  Each step in the optimization takes steps at least this big. When
  none are possible, registration is complete.
  flag: --minimumStepLength %s
movingBinaryVolume: (an existing file name)
  Moving Image binary mask volume, ONLY FOR MANUAL ROI mode.
  flag: --movingBinaryVolume %s

```

```
movingVolume: (an existing file name)
    The moving image for registration by mutual information
    optimization.
    flag: --movingVolume %s
movingVolumeTimeIndex: (an integer (int or long))
    The index in the time series for the 3D moving image to fit, if
    4-dimensional.
    flag: --movingVolumeTimeIndex %d
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels
    flag: --numberOfHistogramBins %d
numberOfIterations: (a list of items which are an integer (int or
    long))
    The maximum number of iterations to try before failing to converge.
    Use an explicit limit like 500 or 1000 to manage risk of divergence
    flag: --numberOfIterations %s
numberOfMatchPoints: (an integer (int or long))
    the number of match points
    flag: --numberOfMatchPoints %d
numberOfSamples: (an integer (int or long))
    The number of voxels sampled for mutual information computation.
    Increase this for a slower, more careful fit. You can also limit the
    sampling focus with ROI masks and ROIAUTO mask generation.
    flag: --numberOfSamples %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use. (default is
    auto-detected)
    flag: --numberOfThreads %d
outputFixedVolumeROI: (a boolean or a file name)
    The ROI automatically found in fixed image, ONLY FOR ROIAUTO mode.
    flag: --outputFixedVolumeROI %s
outputMovingVolumeROI: (a boolean or a file name)
    The ROI automatically found in moving image, ONLY FOR ROIAUTO mode.
    flag: --outputMovingVolumeROI %s
outputTransform: (a boolean or a file name)
    (optional) Filename to which save the (optional) estimated
    transform. NOTE: You must select either the outputTransform or the
    outputVolume option.
    flag: --outputTransform %s
outputVolume: (a boolean or a file name)
    (optional) Output image for registration. NOTE: You must select
    either the outputTransform or the outputVolume option.
    flag: --outputVolume %s
outputVolumePixelType: ('float' or 'short' or 'ushort' or 'int' or
    'uint' or 'uchar')
    The output image Pixel Type is the scalar datatype for
    representation of the Output Volume.
    flag: --outputVolumePixelType %s
permitParameterVariation: (a list of items which are an integer (int
    or long))
    A bit vector to permit linear transform parameters to vary under
    optimization. The vector order corresponds with transform
    parameters, and beyond the end ones fill in as a default. For
    instance, you can choose to rotate only in x (pitch) with 1,0,0;
    this is mostly for expert use in turning on and off individual
    degrees of freedom in rotation, translation or scaling without
    multiplying the number of transform representations; this trick is
    probably meaningless when tried with the general affine transform.
```

```

    flag: --permitParameterVariation %s
projectedGradientTolerance: (a float)
    From itkLBFGSBOptimizer.h: Set/Get the ProjectedGradientTolerance.
    Algorithm terminates when the project gradient is below the
    tolerance. Default lbfgsb value is 1e-5, but 1e-4 seems to work
    well.,
    flag: --projectedGradientTolerance %f
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    flag: --promptUser
relaxationFactor: (a float)
    Internal debugging parameter, and should probably never be used from
    the command line. This will be removed in the future.
    flag: --relaxationFactor %f
removeIntensityOutliers: (a float)
    The half percentage to decide outliers of image intensities. The
    default value is zero, which means no outlier removal. If the value
    of 0.005 is given, the module will throw away 0.005 % of both tails,
    so 0.01% of intensities in total would be ignored in its statistic
    calculation.
    flag: --removeIntensityOutliers %f
reproportionScale: (a float)
    ScaleVersor3D 'Scale' compensation factor. Increase this to put more
    rescaling in a ScaleVersor3D or ScaleSkewVersor3D search pattern.
    1.0 works well with a translationScale of 1000.0
    flag: --reproportionScale %f
scaleOutputValues: (a boolean)
    If true, and the voxel values do not fit within the minimum and
    maximum values of the desired outputVolumePixelType, then linearly
    scale the min/max output image voxel values to fit within the
    min/max range of the outputVolumePixelType.
    flag: --scaleOutputValues
skewScale: (a float)
    ScaleSkewVersor3D Skew compensation factor. Increase this to put
    more skew in a ScaleSkewVersor3D search pattern. 1.0 works well with
    a translationScale of 1000.0
    flag: --skewScale %f
splineGridSize: (a list of items which are an integer (int or long))
    The number of subdivisions of the BSpline Grid to be centered on the
    image space. Each dimension must have at least 3 subdivisions for
    the BSpline to be correctly computed.
    flag: --splineGridSize %s
strippedOutputTransform: (a boolean or a file name)
    File name for the rigid component of the estimated affine transform.
    Can be used to rigidly register the moving image to the fixed image.
    NOTE: This value is overwritten if either bsplineTransform or
    linearTransform is set.
    flag: --strippedOutputTransform %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformType: (a list of items which are a unicode string)
    Specifies a list of registration types to be used. The valid types
    are, Rigid, ScaleVersor3D, ScaleSkewVersor3D, Affine, and BSpline.
    Specifying more than one in a comma separated list will initialize
    the next stage with the previous results. If registrationClass flag

```

```

        is used, it overrides this parameter setting.
        flag: --transformType %s
translationScale: (a float)
    How much to scale up changes in position compared to unit rotational
    changes in radians -- decrease this to put more rotation in the
    search pattern.
    flag: --translationScale %f
useAffine: (a boolean)
    Perform an Affine registration as part of the sequential
    registration steps. This family of options superceeds the use of
    transformType if any of them are set.
    flag: --useAffine
useBSpline: (a boolean)
    Perform a BSpline registration as part of the sequential
    registration steps. This family of options superceeds the use of
    transformType if any of them are set.
    flag: --useBSpline
useCachingOfBSplineWeightsMode: ('ON' or 'OFF')
    This is a 5x speed advantage at the expense of requiring much more
    memory. Only relevant when transformType is BSpline.
    flag: --useCachingOfBSplineWeightsMode %s
useExplicitPDFDerivativesMode: ('AUTO' or 'ON' or 'OFF')
    Using mode AUTO means OFF for BSplineDeformableTransforms and ON for
    the linear transforms. The ON alternative uses more memory to
    sometimes do a better job.
    flag: --useExplicitPDFDerivativesMode %s
useRigid: (a boolean)
    Perform a rigid registration as part of the sequential registration
    steps. This family of options superceeds the use of transformType if
    any of them are set.
    flag: --useRigid
useScaleSkewVersor3D: (a boolean)
    Perform a ScaleSkewVersor3D registration as part of the sequential
    registration steps. This family of options superceeds the use of
    transformType if any of them are set.
    flag: --useScaleSkewVersor3D
useScaleVersor3D: (a boolean)
    Perform a ScaleVersor3D registration as part of the sequential
    registration steps. This family of options superceeds the use of
    transformType if any of them are set.
    flag: --useScaleVersor3D
writeOutputTransformInFloat: (a boolean)
    By default, the output registration transforms (either the output
    composite transform or each transform component) are written to the
    disk in double precision. If this flag is ON, the output transforms
    will be written in single (float) precision. It is especially
    important if the output transform is a displacement field transform,
    or it is a composite transform that includes several displacement
    fields.
    flag: --writeOutputTransformInFloat
writeTransformOnFailure: (a boolean)
    Flag to save the final transform even if the numberOfIterations are
    reached without convergence. (Intended for use when
    --failureExitCode 0 )
    flag: --writeTransformOnFailure

```

Outputs:

```

bsplineTransform: (an existing file name)
    (optional) Filename to which save the estimated transform. NOTE: You
    must set at least one output object (either a deformed image or a
    transform. NOTE: USE THIS ONLY IF THE FINAL TRANSFORM IS BSpline
linearTransform: (an existing file name)
    (optional) Filename to which save the estimated transform. NOTE: You
    must set at least one output object (either a deformed image or a
    transform. NOTE: USE THIS ONLY IF THE FINAL TRANSFORM IS ---NOT---
    BSpline
outputFixedVolumeROI: (an existing file name)
    The ROI automatically found in fixed image, ONLY FOR ROIAUTO mode.
outputMovingVolumeROI: (an existing file name)
    The ROI automatically found in moving image, ONLY FOR ROIAUTO mode.
outputTransform: (an existing file name)
    (optional) Filename to which save the (optional) estimated
    transform. NOTE: You must select either the outputTransform or the
    outputVolume option.
outputVolume: (an existing file name)
    (optional) Output image for registration. NOTE: You must select
    either the outputTransform or the outputVolume option.
strippedOutputTransform: (an existing file name)
    File name for the rigid component of the estimated affine transform.
    Can be used to rigidly register the moving image to the fixed image.
    NOTE: This value is overwritten if either bsplineTransform or
    linearTransform is set.

```

interfaces.slicer.registration.brainsresample

BRAINSResample

[Link to code](#)

Wraps command ****BRAINSResample****

title: Resample Image (BRAINS)

category: Registration

description: This program resamples an image image using a deformation field or a transform (BSpline, Affine, Rigid, etc.).

version: 3.0.0

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Modules:BRAINSResample>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Vincent Magnotta, Greg Harris, and Hans Johnson.

acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
defaultValue: (a float)
    Default voxel value
    flag: --defaultValue %f
deformationVolume: (an existing file name)

```

```

Displacement Field to be used to warp the image
flag: --deformationVolume %s
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a value
of class 'str', nipy default value: {})
Environment variables
gridSpacing: (a list of items which are an integer (int or long))
Add warped grid to output image to help show the deformation that
occured with specified spacing. A spacing of 0 in a dimension
indicates that grid lines should be rendered to fall exactly (i.e.
do not allow displacements off that plane). This is useful for
making a 2D image of grid lines from the 3D space
flag: --gridSpacing %s
ignore_exception: (a boolean, nipy default value: False)
Print an error message instead of throwing an exception in case the
interface fails to run
inputVolume: (an existing file name)
Image To Warp
flag: --inputVolume %s
interpolationMode: ('NearestNeighbor' or 'Linear' or
'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
Type of interpolation to be used when applying transform to moving
volume. Options are Linear, ResampleInPlace, NearestNeighbor,
BSpline, or WindowedSinc
flag: --interpolationMode %s
inverseTransform: (a boolean)
True/False is to compute inverse of given transformation. Default is
false
flag: --inverseTransform
numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
flag: --numberOfThreads %d
outputVolume: (a boolean or a file name)
Resulting deformed image
flag: --outputVolume %s
pixelType: ('float' or 'short' or 'ushort' or 'int' or 'uint' or
'uchar' or 'binary')
Specifies the pixel type for the input/output images. The 'binary'
pixel type uses a modified algorithm whereby the image is read in as
unsigned char, a signed distance map is created, signed distance map
is resampled, and then a thresholded image of type unsigned char is
written to disk.
flag: --pixelType %s
referenceVolume: (an existing file name)
Reference image used only to define the output space. If not
specified, the warping is done in the same space as the image to
warp.
flag: --referenceVolume %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
warpTransform: (an existing file name)
Filename for the BRAINSFit transform used in place of the
deformation field
flag: --warpTransform %s

```

Outputs:


```
outputVolume: (an existing file name)
    Resulting deformed image
```

interfaces.slicer.registration.specialized

ACPCTransform

[Link to code](#)

Wraps command ****ACPCTransform****

title: ACPC Transform

category: Registration.Specialized

description: <p>Calculate a transformation from two lists of fiducial points.</p><p>ACPC line is two fiducial points, one at the anterior commissure and one at the posterior commissure. The resulting transform will bring the line connecting them to horizontal to the AP axis.</p><p>The midline is a series of points defining the division between the hemispheres of the brain (the mid sagittal plane). The resulting transform will put the output volume with the mid sagittal plane lined up with the AS plane.</p><p>Use the Filtering moduleResample Scalar/Vector/DWI Volume to apply the transformation to a volume.</p>

version: 1.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ACPCTransform>

license: slicer3

contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
acpc: (a list of items which are a list of from 3 to 3 items which
    are a float)
    ACPC line, two fiducial points, one at the anterior commissure and
    one at the posterior commissure.
    flag: --acpc %s...
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debugSwitch: (a boolean)
    Click if wish to see debugging output
    flag: --debugSwitch
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
midline: (a list of items which are a list of from 3 to 3 items which
    are a float)
    The midline is a series of points defining the division between the
    hemispheres of the brain (the mid sagittal plane).
    flag: --midline %s...
outputTransform: (a boolean or a file name)
    A transform filled in from the ACPC and Midline registration
    calculation
    flag: --outputTransform %s
```

```
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputTransform: (an existing file name)
    A transform filled in from the ACPC and Midline registration
    calculation
```

BRAINSDemonWarp

[Link to code](#)

Wraps command ****BRAINSDemonWarp****

title: Demon Registration (BRAINS)

category: Registration.Specialized

description: This program finds a deformation field to warp a moving image onto a fixed image. The images must be of the same signal kind, and contain an image of the same kind of object. This program uses the Thirion Demons warp software in ITK, the Insight Toolkit. Additional information is available at: <http://www.nitrc.org/projects/brainsdemonwarp>.

version: 3.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:BRAINSDemonWarp>

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Hans J. Johnson and Greg Harris.

acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
arrayOfPyramidLevelIterations: (a list of items which are an integer
    (int or long))
    The number of iterations for each pyramid level
    flag: --arrayOfPyramidLevelIterations %s
backgroundFillValue: (an integer (int or long))
    Replacement value to overwrite background when performing BOBF
    flag: --backgroundFillValue %d
checkerboardPatternSubdivisions: (a list of items which are an
    integer (int or long))
    Number of Checkerboard subdivisions in all 3 directions
    flag: --checkerboardPatternSubdivisions %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixedBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Fixed image.
    flag: --fixedBinaryVolume %s
fixedVolume: (an existing file name)
    Required: input fixed (target) image
    flag: --fixedVolume %s
```

```

gradient_type: ('0' or '1' or '2')
    Type of gradient used for computing the demons force (0 is
    symmetrized, 1 is fixed image, 2 is moving image)
    flag: --gradient_type %s
gui: (a boolean)
    Display intermediate image volumes for debugging
    flag: --gui
histogramMatch: (a boolean)
    Histogram Match the input images. This is suitable for images of the
    same modality that may have different absolute scales, but the same
    overall intensity profile.
    flag: --histogramMatch
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initializeWithDisplacementField: (an existing file name)
    Initial deformation field vector image file name
    flag: --initializeWithDisplacementField %s
initializeWithTransform: (an existing file name)
    Initial Transform filename
    flag: --initializeWithTransform %s
inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    Input volumes will be typecast to this format:
    float|short|ushort|int|uchar
    flag: --inputPixelType %s
interpolationMode: ('NearestNeighbor' or 'Linear' or
    'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
    'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
    Type of interpolation to be used when applying transform to moving
    volume. Options are Linear, ResampleInPlace, NearestNeighbor,
    BSpline, or WindowedSinc
    flag: --interpolationMode %s
lowerThresholdForBOBF: (an integer (int or long))
    Lower threshold for performing BOBF
    flag: --lowerThresholdForBOBF %d
maskProcessingMode: ('NOMASK' or 'ROIAUTO' or 'ROI' or 'BOBF')
    What mode to use for using the masks: NOMASK|ROIAUTO|ROI|BOBF. If
    ROIAUTO is choosen, then the mask is implicitly defined using a otsu
    foreground and hole filling algorithm. Where the Region Of Interest
    mode uses the masks to define what parts of the image should be used
    for computing the deformation field. Brain Only Background Fill uses
    the masks to pre-process the input images by clipping and filling in
    the background with a predefined value.
    flag: --maskProcessingMode %s
max_step_length: (a float)
    Maximum length of an update vector (0: no restriction)
    flag: --max_step_length %f
medianFilterSize: (a list of items which are an integer (int or
    long))
    Median filter radius in all 3 directions. When images have a lot of
    salt and pepper noise, this step can improve the registration.
    flag: --medianFilterSize %s
minimumFixedPyramid: (a list of items which are an integer (int or
    long))
    The shrink factor for the first level of the fixed image pyramid.
    (i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
    full scale)
    flag: --minimumFixedPyramid %s

```

```
minimumMovingPyramid: (a list of items which are an integer (int or
    long))
    The shrink factor for the first level of the moving image pyramid.
    (i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
    full scale)
    flag: --minimumMovingPyramid %s
movingBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Moving image.
    flag: --movingBinaryVolume %s
movingVolume: (an existing file name)
    Required: input moving image
    flag: --movingVolume %s
neighborhoodForBOBF: (a list of items which are an integer (int or
    long))
    neighborhood in all 3 directions to be included when performing BOBF
    flag: --neighborhoodForBOBF %s
numberOfBCHApproximationTerms: (an integer (int or long))
    Number of terms in the BCH expansion
    flag: --numberOfBCHApproximationTerms %d
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels
    flag: --numberOfHistogramBins %d
numberOfMatchPoints: (an integer (int or long))
    The number of match points for histogramMatch
    flag: --numberOfMatchPoints %d
numberOfPyramidLevels: (an integer (int or long))
    Number of image pyramid levels to use in the multi-resolution
    registration.
    flag: --numberOfPyramidLevels %d
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
outputCheckerboardVolume: (a boolean or a file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
    flag: --outputCheckerboardVolume %s
outputDebug: (a boolean)
    Flag to write debugging images after each step.
    flag: --outputDebug
outputDisplacementFieldPrefix: (a unicode string)
    Displacement field filename prefix for writing separate x, y, and z
    component images
    flag: --outputDisplacementFieldPrefix %s
outputDisplacementFieldVolume: (a boolean or a file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
    flag: --outputDisplacementFieldVolume %s
outputNormalized: (a boolean)
    Flag to warp and write the normalized images to output. In
    normalized images the image values are fit-scaled to be between 0
    and the maximum storage type value.
    flag: --outputNormalized
outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    outputVolume will be typecast to this format:
    float|short|ushort|int|uchar
    flag: --outputPixelType %s
outputVolume: (a boolean or a file name)
    Required: output resampled moving image (will have the same physical
```

```

        space as the fixedVolume).
        flag: --outputVolume %s
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    flag: --promptUser
registrationFilterType: ('Demons' or 'FastSymmetricForces' or
    'Diffeomorphic')
    Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic
    flag: --registrationFilterType %s
seedForBOBF: (a list of items which are an integer (int or long))
    coordinates in all 3 directions for Seed when performing BOBF
    flag: --seedForBOBF %s
smoothDisplacementFieldSigma: (a float)
    A gaussian smoothing value to be applied to the deformation feild at
    each iteration.
    flag: --smoothDisplacementFieldSigma %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
upFieldSmoothing: (a float)
    Smoothing sigma for the update field at each iteration
    flag: --upFieldSmoothing %f
upperThresholdForBOBF: (an integer (int or long))
    Upper threshold for performing BOBF
    flag: --upperThresholdForBOBF %d
use_vanilla_dem: (a boolean)
    Run vanilla demons algorithm
    flag: --use_vanilla_dem

```

Outputs:

```

outputCheckerboardVolume: (an existing file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
outputDisplacementFieldVolume: (an existing file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
outputVolume: (an existing file name)
    Required: output resampled moving image (will have the same physical
    space as the fixedVolume).

```

FiducialRegistration[Link to code](#)Wraps command ****FiducialRegistration****

title: Fiducial Registration

category: Registration.Specialized

description: Computes a rigid, similarity or affine transform from a matched list of fiducials

version: 0.1.0.\$Revision\$

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/TransformFromFiducials>

contributor: Casey B Goodlett (Kitware), Dominik Meier (SPL, BWH)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipype default value: {})
    Environment variables
fixedLandmarks: (a list of items which are a list of from 3 to 3
    items which are a float)
    Ordered list of landmarks in the fixed image
    flag: --fixedLandmarks %s...
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
movingLandmarks: (a list of items which are a list of from 3 to 3
    items which are a float)
    Ordered list of landmarks in the moving image
    flag: --movingLandmarks %s...
outputMessage: (a unicode string)
    Provides more information on the output
    flag: --outputMessage %s
rms: (a float)
    Display RMS Error.
    flag: --rms %f
saveTransform: (a boolean or a file name)
    Save the transform that results from registration
    flag: --saveTransform %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transformType: ('Translation' or 'Rigid' or 'Similarity')
    Type of transform to produce
    flag: --transformType %s

```

Outputs:

```

saveTransform: (an existing file name)
    Save the transform that results from registration

```

VBRAINSDemonWarp[Link to code](#)Wraps command ****VBRAINSDemonWarp****

title: Vector Demon Registration (BRAINS)

category: Registration.Specialized

description: This program finds a deformation field to warp a moving image onto a fixed image. The images must be of the same signal kind, and contain an image of the same kind of object. This program uses the Thirion Demons warp software in ITK, the Insight Toolkit. Additional information is available at: <http://www.nitrc.org/projects/brainsdemonwarp>.

version: 3.0.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Modules:BRANDemonWarp>license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>

contributor: This tool was developed by Hans J. Johnson and Greg Harris.

acknowledgements: The development of this tool was supported by funding from grants NS050568 and NS40068 from the National Institute of Neurological Disorders and Stroke and grants MH31593, MH40856, from the National Institute of Mental Health.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
arrayOfPyramidLevelIterations: (a list of items which are an integer
    (int or long))
    The number of iterations for each pyramid level
    flag: --arrayOfPyramidLevelIterations %s
backgroundFillValue: (an integer (int or long))
    Replacement value to overwrite background when performing BOBF
    flag: --backgroundFillValue %d
checkerboardPatternSubdivisions: (a list of items which are an
    integer (int or long))
    Number of Checkerboard subdivisions in all 3 directions
    flag: --checkerboardPatternSubdivisions %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fixedBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Fixed image.
    flag: --fixedBinaryVolume %s
fixedVolume: (a list of items which are an existing file name)
    Required: input fixed (target) image
    flag: --fixedVolume %s...
gradient_type: ('0' or '1' or '2')
    Type of gradient used for computing the demons force (0 is
    symmetrized, 1 is fixed image, 2 is moving image)
    flag: --gradient_type %s
gui: (a boolean)
    Display intermediate image volumes for debugging
    flag: --gui
histogramMatch: (a boolean)
    Histogram Match the input images. This is suitable for images of the
    same modality that may have different absolute scales, but the same
    overall intensity profile.
    flag: --histogramMatch
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
initializeWithDisplacementField: (an existing file name)
    Initial deformation field vector image file name
    flag: --initializeWithDisplacementField %s
initializeWithTransform: (an existing file name)
    Initial Transform filename
    flag: --initializeWithTransform %s
inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    Input volumes will be typecast to this format:
    float|short|ushort|int|uchar
    flag: --inputPixelType %s
interpolationMode: ('NearestNeighbor' or 'Linear' or
```

```
'ResampleInPlace' or 'BSpline' or 'WindowedSinc' or 'Hamming' or
'Cosine' or 'Welch' or 'Lanczos' or 'Blackman')
Type of interpolation to be used when applying transform to moving
volume. Options are Linear, ResampleInPlace, NearestNeighbor,
BSpline, or WindowedSinc
flag: --interpolationMode %s
lowerThresholdForBOBF: (an integer (int or long))
Lower threshold for performing BOBF
flag: --lowerThresholdForBOBF %d
makeBOBF: (a boolean)
Flag to make Brain-Only Background-Filled versions of the input and
target volumes.
flag: --makeBOBF
max_step_length: (a float)
Maximum length of an update vector (0: no restriction)
flag: --max_step_length %f
medianFilterSize: (a list of items which are an integer (int or
long))
Median filter radius in all 3 directions. When images have a lot of
salt and pepper noise, this step can improve the registration.
flag: --medianFilterSize %s
minimumFixedPyramid: (a list of items which are an integer (int or
long))
The shrink factor for the first level of the fixed image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
full scale)
flag: --minimumFixedPyramid %s
minimumMovingPyramid: (a list of items which are an integer (int or
long))
The shrink factor for the first level of the moving image pyramid.
(i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
full scale)
flag: --minimumMovingPyramid %s
movingBinaryVolume: (an existing file name)
Mask filename for desired region of interest in the Moving image.
flag: --movingBinaryVolume %s
movingVolume: (a list of items which are an existing file name)
Required: input moving image
flag: --movingVolume %s...
neighborhoodForBOBF: (a list of items which are an integer (int or
long))
neighborhood in all 3 directions to be included when performing BOBF
flag: --neighborhoodForBOBF %s
numberOfBCHApproximationTerms: (an integer (int or long))
Number of terms in the BCH expansion
flag: --numberOfBCHApproximationTerms %d
numberOfHistogramBins: (an integer (int or long))
The number of histogram levels
flag: --numberOfHistogramBins %d
numberOfMatchPoints: (an integer (int or long))
The number of match points for histogramMatch
flag: --numberOfMatchPoints %d
numberOfPyramidLevels: (an integer (int or long))
Number of image pyramid levels to use in the multi-resolution
registration.
flag: --numberOfPyramidLevels %d
numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use.
```



```

    flag: --numberOfThreads %d
outputCheckerboardVolume: (a boolean or a file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
    flag: --outputCheckerboardVolume %s
outputDebug: (a boolean)
    Flag to write debugging images after each step.
    flag: --outputDebug
outputDisplacementFieldPrefix: (a unicode string)
    Displacement field filename prefix for writing separate x, y, and z
    component images
    flag: --outputDisplacementFieldPrefix %s
outputDisplacementFieldVolume: (a boolean or a file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
    flag: --outputDisplacementFieldVolume %s
outputNormalized: (a boolean)
    Flag to warp and write the normalized images to output. In
    normalized images the image values are fit-scaled to be between 0
    and the maximum storage type value.
    flag: --outputNormalized
outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    outputVolume will be typecast to this format:
    float|short|ushort|int|uchar
    flag: --outputPixelType %s
outputVolume: (a boolean or a file name)
    Required: output resampled moving image (will have the same physical
    space as the fixedVolume).
    flag: --outputVolume %s
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    flag: --promptUser
registrationFilterType: ('Demons' or 'FastSymmetricForces' or
    'Diffeomorphic' or 'LogDemons' or 'SymmetricLogDemons')
    Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic|L
    ogDemons|SymmetricLogDemons
    flag: --registrationFilterType %s
seedForBOBF: (a list of items which are an integer (int or long))
    coordinates in all 3 directions for Seed when performing BOBF
    flag: --seedForBOBF %s
smoothDisplacementFieldSigma: (a float)
    A gaussian smoothing value to be applied to the deformation feild at
    each iteration.
    flag: --smoothDisplacementFieldSigma %f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
upFieldSmoothing: (a float)
    Smoothing sigma for the update field at each iteration
    flag: --upFieldSmoothing %f
upperThresholdForBOBF: (an integer (int or long))
    Upper threshold for performing BOBF
    flag: --upperThresholdForBOBF %d
use_vanilla_dem: (a boolean)
    Run vanilla demons algorithm
    flag: --use_vanilla_dem

```

```
weightFactors: (a list of items which are a float)
    Weight factors for each input images
    flag: --weightFactors %s
```

Outputs:

```
outputCheckerboardVolume: (an existing file name)
    Generate a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
outputDisplacementFieldVolume: (an existing file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
outputVolume: (an existing file name)
    Required: output resampled moving image (will have the same physical
    space as the fixedVolume).
```

interfaces.slicer.segmentation.simpleregiongrowingsegmentation

SimpleRegionGrowingSegmentation

[Link to code](#)

Wraps command ****SimpleRegionGrowingSegmentation****

title: Simple Region Growing Segmentation

category: Segmentation

description: A simple region growing segmentation algorithm based on intensity statistics. To create a list of fiducials (Seeds) for this algorithm, click on the tool bar icon of an arrow pointing to a starburst fiducial to enter the 'place a new object mode' and then use the fiducials module. This module uses the Slicer Command Line Interface (CLI) and the ITK filters CurvatureFlowImageFilter and ConfidenceConnectedImageFilter.

version: 0.1.0.\$Revision: 19904 \$(alpha)

documentation-url: <http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/SimpleRegionGrowingSegmentation>

contributor: Jim Miller (GE)

acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software Consortium

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input volume to be filtered
    flag: %s, position: -2
iterations: (an integer (int or long))
    Number of iterations of region growing
    flag: --iterations %d
labelvalue: (an integer (int or long))
```

```

    The integer value (0-255) to use for the segmentation results. This
    will determine the color of the segmentation that will be generated
    by the Region growing algorithm
    flag: --labelvalue %d
multiplier: (a float)
    Number of standard deviations to include in intensity model
    flag: --multiplier %f
neighborhood: (an integer (int or long))
    The radius of the neighborhood over which to calculate intensity
    model
    flag: --neighborhood %d
outputVolume: (a boolean or a file name)
    Output filtered
    flag: %s, position: -1
seed: (a list of items which are a list of from 3 to 3 items which
    are a float)
    Seed point(s) for region growing
    flag: --seed %s...
smoothingIterations: (an integer (int or long))
    Number of smoothing iterations
    flag: --smoothingIterations %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
timestep: (a float)
    Timestep for curvature flow
    flag: --timestep %f

```

Outputs:

```

outputVolume: (an existing file name)
    Output filtered

```

interfaces.slicer.segmentation.specialized

BRAINSROIAuto

[Link to code](#)Wraps command ****BRAINSROIAuto****

title: Foreground masking (BRAINS)

category: Segmentation.Specialized

description: This tool uses a combination of otsu thresholding and a closing operations to identify the most prominent foreground region in an image.

version: 2.4.1

license: <https://www.nitrc.org/svn/brains/BuildScripts/trunk/License.txt>contributor: Hans J. Johnson, hans-johnson -at- uiowa.edu, <http://www.psychiatry.uiowa.edu>

acknowledgements: Hans Johnson(1,3,4); Kent Williams(1); Gregory Harris(1), Vincent Magnotta(1,2,3); Andriy Fedorov(5), fedorov -at- bwh.harvard.edu (Slicer integration); (1=University of Iowa Department of Psychiatry, 2=University of Iowa Department of Radiology, 3=University of Iowa Department of Biomedical Engineering, 4=University of Iowa Department of Electrical and Computer Engineering, 5=Surgical Planning Lab, Harvard)

Inputs:

[Mandatory]

```

[Optional]
ROIAutoDilateSize: (a float)
    This flag is only relevant when using ROIAUTO mode for initializing
    masks. It defines the final dilation size to capture a bit of
    background outside the tissue region. At setting of 10mm has been
    shown to help regularize a BSpline registration type so that there
    is some background constraints to match the edges of the head
    better.
    flag: --ROIAutoDilateSize %f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
closingSize: (a float)
    The Closing Size (in millimeters) for largest connected filled mask.
    This value is divided by image spacing and rounded to the next
    largest voxel number.
    flag: --closingSize %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    The input image for finding the largest region filled mask.
    flag: --inputVolume %s
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    flag: --numberOfThreads %d
otsuPercentileThreshold: (a float)
    Parameter to the Otsu threshold algorithm.
    flag: --otsuPercentileThreshold %f
outputClippedVolumeROI: (a boolean or a file name)
    The inputVolume clipped to the region of the brain mask.
    flag: --outputClippedVolumeROI %s
outputROIMaskVolume: (a boolean or a file name)
    The ROI automatically found from the input image.
    flag: --outputROIMaskVolume %s
outputVolumePixelType: ('float' or 'short' or 'ushort' or 'int' or
    'uint' or 'uchar')
    The output image Pixel Type is the scalar datatype for
    representation of the Output Volume.
    flag: --outputVolumePixelType %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
thresholdCorrectionFactor: (a float)
    A factor to scale the Otsu algorithm's result threshold, in case
    clipping mangles the image.
    flag: --thresholdCorrectionFactor %f

```

Outputs:

```

outputClippedVolumeROI: (an existing file name)
    The inputVolume clipped to the region of the brain mask.
outputROIMaskVolume: (an existing file name)

```

The ROI automatically found **from the input** image.

EMSegmentCommandLine

[Link to code](#)

Wraps command ****EMSegmentCommandLine****

title: EMSegment Command-line

category: Segmentation.Specialized

description: This module is used to simplify the process of segmenting large collections of images by providing a command line interface to the EMSegment algorithm for script and batch processing.

documentation-url: http://www.slicer.org/slicerWiki/index.php/Documentation/4.0/EMSegment_Command-line

contributor: Sebastien Barre, Brad Davis, Kilian Pohl, Polina Golland, Yumin Yuan, Daniel Haehn

acknowledgements: Many people and organizations have contributed to the funding, design, and development of the EMSegment algorithm and its various implementations.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
atlasVolumeFileNames: (a list of items which are an existing file
    name)
    Use an alternative atlas to the one that is specified by the mrml
    file - note the order matters !
    flag: --atlasVolumeFileNames %s...
disableCompression: (a boolean)
    Don't use compression when writing result image to disk.
    flag: --disableCompression
disableMultithreading: (an integer (int or long))
    Disable multithreading for the EMSegmenter algorithm only!
    Preprocessing might still run in multi-threaded mode. -1: Do not
    overwrite default value. 0: Disable. 1: Enable.
    flag: --disableMultithreading %d
dontUpdateIntermediateData: (an integer (int or long))
    Disable update of intermediate results. -1: Do not overwrite default
    value. 0: Disable. 1: Enable.
    flag: --dontUpdateIntermediateData %d
dontWriteResults: (a boolean)
    Used for testing. Don't actually write the resulting labelmap to
    disk.
    flag: --dontWriteResults
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
generateEmptyMRMLSceneAndQuit: (a boolean or a file name)
    Used for testing. Only write a scene with default mrml parameters.
    flag: --generateEmptyMRMLSceneAndQuit %s
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
intermediateResultsDirectory: (an existing directory name)
    Directory where EMSegmenter will write intermediate data (e.g.,
```

```
    aligned atlas data).
    flag: --intermediateResultsDirectory %s
keepTempFiles: (a boolean)
    If flag is set then at the end of command the temporary files are
    not removed
    flag: --keepTempFiles
loadAtlasNonCentered: (a boolean)
    Read atlas files non-centered.
    flag: --loadAtlasNonCentered
loadTargetCentered: (a boolean)
    Read target files centered.
    flag: --loadTargetCentered
mrmlSceneFileName: (an existing file name)
    Active MRML scene that contains EMSegment algorithm parameters.
    flag: --mrmlSceneFileName %s
parametersMRMLNodeName: (a unicode string)
    The name of the EMSegment parameters node within the active MRML
    scene. Leave blank for default.
    flag: --parametersMRMLNodeName %s
registrationAffineType: (an integer (int or long))
    specify the accuracy of the affine registration. -2: Do not
    overwrite default, -1: Test, 0: Disable, 1: Fast, 2: Accurate
    flag: --registrationAffineType %d
registrationDeformableType: (an integer (int or long))
    specify the accuracy of the deformable registration. -2: Do not
    overwrite default, -1: Test, 0: Disable, 1: Fast, 2: Accurate
    flag: --registrationDeformableType %d
registrationPackage: (a unicode string)
    specify the registration package for preprocessing (CMTK or BRAINS
    or PLASTIMATCH or DEMONS)
    flag: --registrationPackage %s
resultMRMLSceneFileName: (a boolean or a file name)
    Write out the MRML scene after command line substitutions have been
    made.
    flag: --resultMRMLSceneFileName %s
resultStandardVolumeFileName: (an existing file name)
    Used for testing. Compare segmentation results to this image and
    return EXIT_FAILURE if they do not match.
    flag: --resultStandardVolumeFileName %s
resultVolumeFileName: (a boolean or a file name)
    The file name that the segmentation result volume will be written
    to.
    flag: --resultVolumeFileName %s
targetVolumeFileNames: (a list of items which are an existing file
    name)
    File names of target volumes (to be segmented). The number of target
    images must be equal to the number of target images specified in the
    parameter set, and these images must be spatially aligned.
    flag: --targetVolumeFileNames %s...
taskPreProcessingSetting: (a unicode string)
    Specifies the different task parameter. Leave blank for default.
    flag: --taskPreProcessingSetting %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean)
    Enable verbose output.
```

```
flag: --verbose
```

Outputs:

```
generateEmptyMRMLSceneAndQuit: (an existing file name)
    Used for testing. Only write a scene with default mrml parameters.
resultMRMLSceneFileName: (an existing file name)
    Write out the MRML scene after command line substitutions have been
    made.
resultVolumeFileName: (an existing file name)
    The file name that the segmentation result volume will be written
    to.
```

RobustStatisticsSegmenter

[Link to code](#)

Wraps command ****RobustStatisticsSegmenter****

title: Robust Statistics Segmenter

category: Segmentation.Specialized

description: Active contour segmentation using robust statistic.

version: 1.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/RobustStatisticsSegmenter>

contributor: Yi Gao (gatech), Allen Tannenbaum (gatech), Ron Kikinis (SPL, BWH)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
curvatureWeight: (a float)
    Given sphere 1.0 score and extreme rough bounday/surface 0 score,
    what is the expected smoothness of the object?
    flag: --curvatureWeight %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
expectedVolume: (a float)
    The approximate volume of the object, in mL.
    flag: --expectedVolume %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
intensityHomogeneity: (a float)
    What is the homogeneity of intensity within the object? Given
    constant intensity at 1.0 score and extreme fluctuating intensity at
    0.
    flag: --intensityHomogeneity %f
labelImageFileName: (an existing file name)
    Label image for initialization
    flag: %s, position: -2
labelValue: (an integer (int or long))
```

```

    Label value of the output image
    flag: --labelValue %d
maxRunningTime: (a float)
    The program will stop if this time is reached.
    flag: --maxRunningTime %f
originalImageFileName: (an existing file name)
    Original image to be segmented
    flag: %s, position: -3
segmentedImageFileName: (a boolean or a file name)
    Segmented image
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

segmentedImageFileName: (an existing file name)
    Segmented image

```

interfaces.slicer.surface

GrayscaleModelMaker

[Link to code](#)Wraps command ****GrayscaleModelMaker****

title: Grayscale Model Maker

category: Surface Models

description: Create 3D surface models from grayscale data. This module uses Marching Cubes to create an isosurface at a given threshold. The resulting surface consists of triangles that separate a volume into regions below and above the threshold. The resulting surface can be smoothed and decimated. This model works on continuous data while the module Model Maker works on labeled (or discrete) data.

version: 3.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/GrayscaleModelMaker>

license: slicer3

contributor: Nicole Aucoin (SPL, BWH), Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
InputVolume: (an existing file name)
    Volume containing the input grayscale data.
    flag: %s, position: -2
OutputGeometry: (a boolean or a file name)
    Output that contains geometry model.
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
decimate: (a float)
    Target reduction during decimation, as a decimal percentage

```



```

    reduction in the number of polygons. If 0, no decimation will be
    done.
    flag: --decimate %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
name: (a unicode string)
    Name to use for this model.
    flag: --name %s
pointnormals: (a boolean)
    Calculate the point normals? Calculated point normals make the
    surface appear smooth. Without point normals, the surface will
    appear faceted.
    flag: --pointnormals
smooth: (an integer (int or long))
    Number of smoothing iterations. If 0, no smoothing will be done.
    flag: --smooth %d
splitnormals: (a boolean)
    Splitting normals is useful for visualizing sharp features. However
    it creates holes in surfaces which affect measurements
    flag: --splitnormals
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
threshold: (a float)
    Grayscale threshold of isosurface. The resulting surface of
    triangles separates the volume into voxels that lie above (inside)
    and below (outside) the threshold.
    flag: --threshold %f

```

Outputs:

```

OutputGeometry: (an existing file name)
    Output that contains geometry model.

```

LabelMapSmoothing

[Link to code](#)Wraps command ****LabelMapSmoothing****

title: Label Map Smoothing

category: Surface Models

description: This filter smoothes a binary label map. With a label map as input, this filter runs an anti-aliasing algorithm followed by a Gaussian smoothing algorithm. The output is a smoothed label map.

version: 1.0

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/LabelMapSmoothing>

contributor: Dirk Padfield (GE), Josh Cates (Utah), Ross Whitaker (Utah)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149. This filter is based on work developed at the University of Utah, and implemented at GE Research.

Inputs:

```

[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
gaussianSigma: (a float)
    The standard deviation of the Gaussian kernel
    flag: --gaussianSigma %f
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputVolume: (an existing file name)
    Input label map to smooth
    flag: %s, position: -2
labelToSmooth: (an integer (int or long))
    The label to smooth. All others will be ignored. If no label is
    selected by the user, the maximum label in the image is chosen by
    default.
    flag: --labelToSmooth %d
maxRMSError: (a float)
    The maximum RMS error.
    flag: --maxRMSError %f
numberOfIterations: (an integer (int or long))
    The number of iterations of the level set AntiAliasing algorithm
    flag: --numberOfIterations %d
outputVolume: (a boolean or a file name)
    Smoothed label map
    flag: %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

outputVolume: (an existing file name)
    Smoothed label map

```

MergeModels[Link to code](#)Wraps command ****MergeModels****

title: Merge Models

category: Surface Models

description: Merge the polydata from two input models and output a new model with the added polydata. Uses the vtkAppendPolyData filter. Works on .vtp and .vtk surface files.

version: \$Revision\$

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/MergeModels>

contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH), Daniel Haehn (SPL, BWH)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
Model1: (an existing file name)
    Model
    flag: %s, position: -3
Model2: (an existing file name)
    Model
    flag: %s, position: -2
ModelOutput: (a boolean or a file name)
    Model
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipytype default value: {})
    Environment variables
ignore_exception: (a boolean, nipytype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
ModelOutput: (an existing file name)
    Model
```

ModelMaker

Link to code

Wraps command ****ModelMaker****

title: Model Maker

category: Surface Models

description: Create 3D surface models from segmented data.<p>Models are imported into Slicer under a model hierarchy node in a MRML scene. The model colors are set by the color table associated with the input volume (these colours will only be visible if you load the model scene file).</p><p>Create Multiple:</p><p>If you specify a list of Labels, it will over ride any start/end label settings.</p><p>If you click<i>Generate All</i>it will over ride the list of lables and any start/end label settings.</p><p>Model Maker Settings:</p><p>You can set the number of smoothing iterations, target reduction in number of polygons (decimal percentage). Use 0 and 1 if you wish no smoothing nor decimation.
You can set the flags to split normals or generate point normals in this pane as well.
You can save a copy of the models after intermediate steps (marching cubes, smoothing, and decimation if not joint smoothing, otherwise just after decimation); these models are not saved in the mrml file, turn off deleting temporary files first in the python window:
<i>slicer.modules.modelmaker.cliModuleLogic().DeleteTemporaryFilesOff()</i></p><p>version: 4.1

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ModelMaker>

license: slicer4

contributor: Nicole Aucoin (SPL, BWH), Ron Kikinis (SPL, BWH), Bill Lorensen (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```

[Mandatory]

[Optional]
InputVolume: (an existing file name)
    Input label map. The Input Volume drop down menu is populated with
    the label map volumes that are present in the scene, select one from
    which to generate models.
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
color: (an existing file name)
    Color table to make labels to colors and objects
    flag: --color %s
debug: (a boolean)
    turn this flag on in order to see debugging output (look in the
    Error Log window that is accessed via the View menu)
    flag: --debug
decimate: (a float)
    Chose the target reduction in number of polygons as a decimal
    percentage (between 0 and 1) of the number of polygons. Specifies
    the percentage of triangles to be removed. For example, 0.1 means
    10% reduction and 0.9 means 90% reduction.
    flag: --decimate %f
end: (an integer (int or long))
    If you want to specify a continuous range of labels from which to
    generate models, enter the higher label here. Voxel value up to
    which to continue making models. Skip any values with zero voxels.
    flag: --end %d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
filtertype: ('Sinc' or 'Laplacian')
    You can control the type of smoothing done on the models by
    selecting a filter type of either Sinc or Laplacian.
    flag: --filtertype %s
generateAll: (a boolean)
    Generate models for all labels in the input volume. select this
    option if you want to create all models that correspond to all
    values in a labelmap volume (using the Joint Smoothing option below
    is useful with this option). Ignores Labels, Start Label, End Label
    settings. Skips label 0.
    flag: --generateAll
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
jointsmooth: (a boolean)
    This will ensure that all resulting models fit together smoothly,
    like jigsaw puzzle pieces. Otherwise the models will be smoothed
    independently and may overlap.
    flag: --jointsmooth
labels: (a list of items which are an integer (int or long))
    A comma separated list of label values from which to make models. f
    you specify a list of Labels, it will override any start/end label
    settings. If you click Generate All Models it will override the list
    of labels and any start/end label settings.
    flag: --labels %s

```

```

modelSceneFile: (a boolean or a list of items which are a file name)
    Generated models, under a model hierarchy node. Models are imported
    into Slicer under a model hierarchy node, and their colors are set
    by the color table associated with the input label map volume. The
    model hierarchy node must be created before running the model maker,
    by selecting Create New ModelHierarchy from the Models drop down
    menu. If you're running from the command line, a model hierarchy
    node in a new mrml scene will be created for you.
    flag: --modelSceneFile %s...
name: (a unicode string)
    Name to use for this model. Any text entered in the entry box will
    be the starting string for the created model file names. The label
    number and the color name will also be part of the file name. If
    making multiple models, use this as a prefix to the label and color
    name.
    flag: --name %s
pad: (a boolean)
    Pad the input volume with zero value voxels on all 6 faces in order
    to ensure the production of closed surfaces. Sets the origin
    translation and extent translation so that the models still line up
    with the unpadded input volume.
    flag: --pad
pointnormals: (a boolean)
    Turn this flag on if you wish to calculate the normal vectors for
    the points.
    flag: --pointnormals
saveIntermediateModels: (a boolean)
    You can save a copy of the models after each of the intermediate
    steps (marching cubes, smoothing, and decimation if not joint
    smoothing, otherwise just after decimation). These intermediate
    models are not saved in the mrml file, you have to load them
    manually after turning off deleting temporary files in they python
    console (View ->Python Interactor) using the following command slice
    r.modules.modelmaker.cliModuleLogic().DeleteTemporaryFilesOff().
    flag: --saveIntermediateModels
skipUnNamed: (a boolean)
    Select this to not generate models from labels that do not have
    names defined in the color look up table associated with the input
    label map. If true, only models which have an entry in the color
    table will be generated. If false, generate all models that exist
    within the label range.
    flag: --skipUnNamed
smooth: (an integer (int or long))
    Here you can set the number of smoothing iterations for Laplacian
    smoothing, or the degree of the polynomial approximating the
    windowed Sinc function. Use 0 if you wish no smoothing.
    flag: --smooth %d
splitnormals: (a boolean)
    Splitting normals is useful for visualizing sharp features. However
    it creates holes in surfaces which affects measurements.
    flag: --splitnormals
start: (an integer (int or long))
    If you want to specify a continuous range of labels from which to
    generate models, enter the lower label here. Voxel value from which
    to start making models. Used instead of the label list to specify a
    range (make sure the label list is empty or it will over ride this).
    flag: --start %d
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')

```

```
Control terminal output: `stream` - displays to terminal immediately
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
modelSceneFile: (a list of items which are an existing file name)
Generated models, under a model hierarchy node. Models are imported
into Slicer under a model hierarchy node, and their colors are set
by the color table associated with the input label map volume. The
model hierarchy node must be created before running the model maker,
by selecting Create New ModelHierarchy from the Models drop down
menu. If you're running from the command line, a model hierarchy
node in a new mrml scene will be created for you.
```

ModelToLabelMap

[Link to code](#)

Wraps command ****ModelToLabelMap****

title: Model To Label Map

category: Surface Models

description: Intersects an input model with an reference volume and produces an output label map.

version: 0.1.0.\$Revision: 8643 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/PolyDataToLabelMap>

contributor: Nicole Aucoin (SPL, BWH), Xiaodong Tao (GE)

acknowledgements: This work is part of the National Alliance for Medical Image Computing (NAMIC), funded by the National Institutes of Health through the NIH Roadmap for Medical Research, Grant U54 EB005149.

Inputs:

```
[Mandatory]

[Optional]
InputVolume: (an existing file name)
    Input volume
    flag: %s, position: -3
OutputVolume: (a boolean or a file name)
    The label volume
    flag: %s, position: -1
args: (a unicode string)
    Additional parameters to the command
    flag: %s
distance: (a float)
    Sample distance
    flag: --distance %f
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
surface: (an existing file name)
    Model
    flag: %s, position: -2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
```

```
(default), `allatonce` - waits till command is finished to display
output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
OutputVolume: (an existing file name)
               The label volume
```

ProbeVolumeWithModel[Link to code](#)Wraps command ****ProbeVolumeWithModel****

title: Probe Volume With Model

category: Surface Models

description: Paint a model by a volume (using vtkProbeFilter).

version: 0.1.0.\$Revision: 1892 \$(alpha)

documentation-url: <http://wiki.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ProbeVolumeWithModel>

contributor: Lauren O'Donnell (SPL, BWH)

acknowledgements: BWH, NCIGT/LMI

Inputs:

```
[Mandatory]

[Optional]
InputModel: (an existing file name)
            Input model
            flag: %s, position: -2
InputVolume: (an existing file name)
             Volume to use to 'paint' the model
             flag: %s, position: -3
OutputModel: (a boolean or a file name)
             Output 'painted' model
             flag: %s, position: -1
args: (a unicode string)
      Additional parameters to the command
      flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipy default value: {})
         Environment variables
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
                  Control terminal output: `stream` - displays to terminal immediately
                  (default), `allatonce` - waits till command is finished to display
                  output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
OutputModel: (an existing file name)
              Output 'painted' model
```

interfaces.slicer.utilities

EMSegmentTransformToNewFormat

[Link to code](#)

Wraps command ****EMSegmentTransformToNewFormat ****

title: Transform MRML Files to New EMSegmenter Standard

category: Utilities

description: Transform MRML Files to New EMSegmenter Standard

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyte default value: {})
    Environment variables
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
inputMRMLFileName: (an existing file name)
    Active MRML scene that contains EMSegment algorithm parameters in
    the format before 3.6.3 - please include absolute file name in path.
    flag: --inputMRMLFileName %s
outputMRMLFileName: (a boolean or a file name)
    Write out the MRML scene after transformation to format 3.6.3 has
    been made. - has to be in the same directory as the input MRML file
    due to Slicer Core bug - please include absolute file name in path
    flag: --outputMRMLFileName %s
templateFlag: (a boolean)
    Set to true if the transformed mrml file should be used as template
    file
    flag: --templateFlag
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
outputMRMLFileName: (an existing file name)
    Write out the MRML scene after transformation to format 3.6.3 has
    been made. - has to be in the same directory as the input MRML file
    due to Slicer Core bug - please include absolute file name in path
```


interfaces.spm.model

EstimateContrast

[Link to code](#)

Use `spm_contrasts` to estimate contrasts of interest

Examples

```
>>> import nipy.interfaces.spm as spm
>>> est = spm.EstimateContrast()
>>> est.inputs.spm_mat_file = 'SPM.mat'
>>> cont1 = ('Task>Baseline', 'T', ['Task-Odd', 'Task-Even'], [0.5, 0.5])
>>> cont2 = ('Task-Odd>Task-Even', 'T', ['Task-Odd', 'Task-Even'], [1, -1])
>>> contrasts = [cont1, cont2]
>>> est.inputs.contrasts = contrasts
>>> est.run()
```

Inputs:

[Mandatory]

`beta_images`: (a `list` of items which are an existing file name)
 Parameter estimates of the design matrix

`contrasts`: (a `list` of items which are a `tuple` of the form: (a unicode string, 'T', a `list` of items which are a unicode string, a `list` of items which are a `float`) **or** a `tuple` of the form: (a unicode string, 'T', a `list` of items which are a unicode string, a `list` of items which are a `float`, a `list` of items which are a `float`) **or** a `tuple` of the form: (a unicode string, 'F', a `list` of items which are a `tuple` of the form: (a unicode string, 'T', a `list` of items which are a unicode string, a `list` of items which are a `float`) **or** a `tuple` of the form: (a unicode string, 'T', a `list` of items which are a unicode string, a `list` of items which are a `float`, a `list` of items which are a `float`)))

List of contrasts **with** each contrast being a `list` of the form:
 [('name', 'stat', [condition list], [weight list], [session list])]

If session list **is None or not** provided, all sessions are used. For

```

        F contrasts, the condition list should contain previously defined
        T-contrasts.
    residual_image: (an existing file name)
        Mean-squared image of the residuals
    spm_mat_file: (an existing file name)
        Absolute path to SPM.mat

[Optional]
    group_contrast: (a boolean)
        higher level contrast
        mutually_exclusive: use_derivs
    ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
    matlab_cmd: (a unicode string)
        matlab command to use
    mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
    paths: (a list of items which are a directory name)
        Paths to add to matlabpath
    use_derivs: (a boolean)
        use derivatives for estimation
        mutually_exclusive: group_contrast
    use_mcr: (a boolean)
        Run m-code using SPM MCR
    use_v8struct: (a boolean, nipy default value: True)
        Generate SPM8 and higher compatible jobs

```

Outputs:

```

    con_images: (a list of items which are an existing file name)
        contrast images from a t-contrast
    ess_images: (a list of items which are an existing file name)
        contrast images from an F-contrast
    spmF_images: (a list of items which are an existing file name)
        stat images from an F-contrast
    spmT_images: (a list of items which are an existing file name)
        stat images from a t-contrast
    spm_mat_file: (an existing file name)
        Updated SPM mat file

```

References:: None

EstimateModel[Link to code](#)Use `spm_spm` to estimate the parameters of a model<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=69>**Examples**

```

>>> est = EstimateModel()
>>> est.inputs.spm_mat_file = 'SPM.mat'
>>> est.inputs.estimation_method = {'Classical': 1}
>>> est.run()

```

Inputs:

```

[Mandatory]
estimation_method: (a dictionary with keys which are 'Classical' or
                    'Bayesian2' or 'Bayesian' and with values which are any value)
                    Dictionary of either Classical: 1, Bayesian: 1, or Bayesian2: 1
                    (dict)
spm_mat_file: (an existing file name)
               Absolute path to SPM.mat

[Optional]
flags: (a dictionary with keys which are any value and with values
        which are any value)
        Additional arguments
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
matlab_cmd: (a unicode string)
             matlab command to use
mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
use_mcr: (a boolean)
          Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
               Generate SPM8 and higher compatible jobs
write_residuals: (a boolean)
                 Write individual residual images

```

Outputs:

```

ARcoef: (a list of items which are an existing file name)
         Images of the AR coefficient
Cbetas: (a list of items which are an existing file name)
         Images of the parameter posteriors
RPVimage: (an existing file name)
           Resels per voxel image
SDbetas: (a list of items which are an existing file name)
          Images of the standard deviation of parameter posteriors
SDerror: (a list of items which are an existing file name)
          Images of the standard deviation of the error
beta_images: (a list of items which are an existing file name)
              design parameter estimates
labels: (an existing file name)
         label file
mask_image: (an existing file name)
             binary mask to constrain estimation
residual_image: (an existing file name)
                Mean-squared image of the residuals
residual_images: (a list of items which are an existing file name)
                  individual residual images (requires `write_residuals`)
spm_mat_file: (an existing file name)
               Updated SPM mat file

```

References:: None

FactorialDesign[Link to code](#)

Base class for factorial designs

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=77>

Inputs:

```
[Mandatory]

[Optional]
covariates: (a list of items which are a dictionary with keys which
            are 'vector' or 'name' or 'interaction' or 'centering' and with
            values which are any value)
            covariate dictionary {vector, name, interaction, centering}
explicit_mask_file: (a file name)
                    use an implicit mask file to threshold
global_calc_mean: (a boolean)
                  use mean for global calculation
                  mutually_exclusive: global_calc_omit, global_calc_values
global_calc_omit: (a boolean)
                  omit global calculation
                  mutually_exclusive: global_calc_mean, global_calc_values
global_calc_values: (a list of items which are a float)
                    omit global calculation
                    mutually_exclusive: global_calc_mean, global_calc_omit
global_normalization: (1 or 2 or 3)
                      global normalization None-1, Proportional-2, ANCOVA-3
ignore_exception: (a boolean, nipyne default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
matlab_cmd: (a unicode string)
            matlab command to use
mfile: (a boolean, nipyne default value: True)
       Run m-code using m-file
no_grand_mean_scaling: (a boolean)
                       do not perform grand mean scaling
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
spm_mat_dir: (an existing directory name)
             directory to store SPM.mat file (opt)
threshold_mask_absolute: (a float)
                         use an absolute threshold
                         mutually_exclusive: threshold_mask_none, threshold_mask_relative
threshold_mask_none: (a boolean)
                     do not use threshold masking
                     mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
threshold_mask_relative: (a float)
                         threshold using a proportion of the global value
                         mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use_implicit_threshold: (a boolean)
                        use implicit mask NaNs or zeros to threshold
use_mcr: (a boolean)
          Run m-code using SPM MCR
use_v8struct: (a boolean, nipyne default value: True)
              Generate SPM8 and higher compatible jobs
```

Outputs:

```
spm_mat_file: (an existing file name)
              SPM mat file
```

References:: None

Level1Design

[Link to code](#)

Generate an SPM design matrix

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=59>

Examples

```
>>> level1design = Level1Design()
>>> level1design.inputs.timing_units = 'secs'
>>> level1design.inputs.interscan_interval = 2.5
>>> level1design.inputs.bases = {'hrf':{'derivs': [0,0]}}
>>> level1design.inputs.session_info = 'session_info.npz'
>>> level1design.run()
```

Inputs:

```
[Mandatory]
bases: (a dictionary with keys which are 'hrf' or 'fourier' or
      'fourier_han' or 'gamma' or 'fir' and with values which are any
      value)
      dict {'name':{'basesparam1':val,...}}
      name : string
      Name of basis function (hrf, fourier, fourier_han,
      gamma, fir)
      hrf :
      derivs : 2-element list
      Model HRF Derivatives. No derivatives: [0,0],
      Time derivatives : [1,0], Time and Dispersion
      derivatives: [1,1]
      fourier, fourier_han, gamma, fir:
      length : int
      Post-stimulus window length (in seconds)
      order : int
      Number of basis functions
interscan_interval: (a float)
      Interscan interval in secs
session_info: (any value)
      Session specific information generated by ``modelgen.SpecifyModel``
timing_units: ('secs' or 'scans')
      units for specification of onsets

[Optional]
factor_info: (a list of items which are a dictionary with keys which
            are 'name' or 'levels' and with values which are any value)
            Factor specific information file (opt)
global_intensity_normalization: ('none' or 'scaling')
            Global intensity normalization - scaling or none
ignore_exception: (a boolean, nipy default value: False)
            Print an error message instead of throwing an exception in case the
            interface fails to run
mask_image: (an existing file name)
            Image for explicitly masking the analysis
mask_threshold: ('-Inf' or a float, nipy default value: -Inf)
            Thresholding for the mask
matlab_cmd: (a unicode string)
            matlab command to use
mfile: (a boolean, nipy default value: True)
```

```

    Run m-code using m-file
microtime_onset: (a float)
    The onset/time-bin in seconds for alignment (opt)
microtime_resolution: (an integer (int or long))
    Number of time-bins per scan in secs (opt)
model_serial_correlations: ('AR(1)' or 'FAST' or 'none')
    Model serial correlations AR(1), FAST or none. FAST is available in
    SPM12
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
spm_mat_dir: (an existing directory name)
    directory to store SPM.mat file (opt)
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
volterra_expansion_order: (1 or 2)
    Model interactions - yes:1, no:2

```

Outputs:

```

spm_mat_file: (an existing file name)
    SPM mat file

```

References:: None

MultipleRegressionDesign[Link to code](#)

Create SPM design for multiple regression

Examples

```

>>> mreg = MultipleRegressionDesign()
>>> mreg.inputs.in_files = ['cont1.nii', 'cont2.nii']
>>> mreg.run()

```

Inputs:

```

[Mandatory]
in_files: (a list of at least 2 items which are an existing file
    name)
    List of files

[Optional]
covariates: (a list of items which are a dictionary with keys which
    are 'vector' or 'name' or 'interaction' or 'centering' and with
    values which are any value)
    covariate dictionary {vector, name, interaction, centering}
explicit_mask_file: (a file name)
    use an implicit mask file to threshold
global_calc_mean: (a boolean)
    use mean for global calculation
    mutually_exclusive: global_calc_omit, global_calc_values
global_calc_omit: (a boolean)
    omit global calculation
    mutually_exclusive: global_calc_mean, global_calc_values
global_calc_values: (a list of items which are a float)

```

```

    omit global calculation
    mutually_exclusive: global_calc_mean, global_calc_omit
global_normalization: (1 or 2 or 3)
    global normalization None-1, Proportional-2, ANCOVA-3
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
include_intercept: (a boolean, nipyre default value: True)
    Include intercept in design
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipyre default value: True)
    Run m-code using m-file
no_grand_mean_scaling: (a boolean)
    do not perform grand mean scaling
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
spm_mat_dir: (an existing directory name)
    directory to store SPM.mat file (opt)
threshold_mask_absolute: (a float)
    use an absolute threshold
    mutually_exclusive: threshold_mask_none, threshold_mask_relative
threshold_mask_none: (a boolean)
    do not use threshold masking
    mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
threshold_mask_relative: (a float)
    threshold using a proportion of the global value
    mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use_implicit_threshold: (a boolean)
    use implicit mask NaNs or zeros to threshold
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipyre default value: True)
    Generate SPM8 and higher compatible jobs
user_covariates: (a list of items which are a dictionary with keys
    which are 'vector' or 'name' or 'centering' and with values which
    are any value)
    covariate dictionary {vector, name, centering}

```

Outputs:

```

spm_mat_file: (an existing file name)
    SPM mat file

```

References:: None

OneSampleTTestDesign

[Link to code](#)

Create SPM design for one sample t-test

Examples

```

>>> ttest = OneSampleTTestDesign()
>>> ttest.inputs.in_files = ['cont1.nii', 'cont2.nii']
>>> ttest.run()

```

Inputs:

```

[Mandatory]
in_files: (a list of at least 2 items which are an existing file
          name)
          input files

[Optional]
covariates: (a list of items which are a dictionary with keys which
            are 'vector' or 'name' or 'interaction' or 'centering' and with
            values which are any value)
            covariate dictionary {vector, name, interaction, centering}
explicit_mask_file: (a file name)
                    use an implicit mask file to threshold
global_calc_mean: (a boolean)
                  use mean for global calculation
                  mutually_exclusive: global_calc_omit, global_calc_values
global_calc_omit: (a boolean)
                  omit global calculation
                  mutually_exclusive: global_calc_mean, global_calc_values
global_calc_values: (a list of items which are a float)
                   omit global calculation
                   mutually_exclusive: global_calc_mean, global_calc_omit
global_normalization: (1 or 2 or 3)
                     global normalization None-1, Proportional-2, ANCOVA-3
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
matlab_cmd: (a unicode string)
            matlab command to use
mfile: (a boolean, nipy default value: True)
       Run m-code using m-file
no_grand_mean_scaling: (a boolean)
                       do not perform grand mean scaling
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
spm_mat_dir: (an existing directory name)
             directory to store SPM.mat file (opt)
threshold_mask_absolute: (a float)
                         use an absolute threshold
                         mutually_exclusive: threshold_mask_none, threshold_mask_relative
threshold_mask_none: (a boolean)
                     do not use threshold masking
                     mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
threshold_mask_relative: (a float)
                         threshold using a proportion of the global value
                         mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use_implicit_threshold: (a boolean)
                        use implicit mask NaNs or zeros to threshold
use_mcr: (a boolean)
          Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
              Generate SPM8 and higher compatible jobs

```

Outputs:

```

spm_mat_file: (an existing file name)
              SPM mat file

```

References:: None

PairedTTestDesign

[Link to code](#)

Create SPM design for paired t-test

Examples

```
>>> pttest = PairedTTestDesign()
>>> pttest.inputs.paired_files = [['cont1.nii', 'cont1a.nii'], ['cont2.nii', 'cont2a.
↪nii']]
>>> pttest.run()
```

Inputs:

```
[Mandatory]
paired_files: (a list of at least 2 items which are a list of from 2
               to 2 items which are an existing file name)
               List of paired files

[Optional]
ancova: (a boolean)
        Specify ancova-by-factor regressors
covariates: (a list of items which are a dictionary with keys which
             are 'vector' or 'name' or 'interaction' or 'centering' and with
             values which are any value)
             covariate dictionary {vector, name, interaction, centering}
explicit_mask_file: (a file name)
                    use an implicit mask file to threshold
global_calc_mean: (a boolean)
                  use mean for global calculation
                  mutually_exclusive: global_calc_omit, global_calc_values
global_calc_omit: (a boolean)
                  omit global calculation
                  mutually_exclusive: global_calc_mean, global_calc_values
global_calc_values: (a list of items which are a float)
                   omit global calculation
                   mutually_exclusive: global_calc_mean, global_calc_omit
global_normalization: (1 or 2 or 3)
                     global normalization None-1, Proportional-2, ANCOVA-3
grand_mean_scaling: (a boolean)
                    Perform grand mean scaling
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
matlab_cmd: (a unicode string)
             matlab command to use
mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
no_grand_mean_scaling: (a boolean)
                       do not perform grand mean scaling
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
spm_mat_dir: (an existing directory name)
             directory to store SPM.mat file (opt)
threshold_mask_absolute: (a float)
                         use an absolute threshold
                         mutually_exclusive: threshold_mask_none, threshold_mask_relative
threshold_mask_none: (a boolean)
                     do not use threshold masking
```

```

    mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
threshold_mask_relative: (a float)
    threshold using a proportion of the global value
    mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use_implicit_threshold: (a boolean)
    use implicit mask NaNs or zeros to threshold
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs

```

Outputs:

```

spm_mat_file: (an existing file name)
    SPM mat file

```

References:: None

Threshold[Link to code](#)

Topological FDR thresholding based on cluster extent/size. Smoothness is estimated from GLM residuals but is assumed to be the same for all of the voxels.

Examples

```

>>> thresh = Threshold()
>>> thresh.inputs.spm_mat_file = 'SPM.mat'
>>> thresh.inputs.stat_image = 'spmT_0001.img'
>>> thresh.inputs.contrast_index = 1
>>> thresh.inputs.extent_fdr_p_threshold = 0.05
>>> thresh.run()

```

Inputs:

```

[Mandatory]
contrast_index: (an integer (int or long))
    which contrast in the SPM.mat to use
spm_mat_file: (an existing file name)
    absolute path to SPM.mat
stat_image: (an existing file name)
    stat image

[Optional]
extent_fdr_p_threshold: (a float, nipy default value: 0.05)
    p threshold on FDR corrected cluster size probabilities
extent_threshold: (an integer (int or long), nipy default value: 0)
    Minimum cluster size in voxels
force_activation: (a boolean, nipy default value: False)
    In case no clusters survive the topological inference step this will
    pick a cluster with the highest sum of t-values. Use with care.
height_threshold: (a float, nipy default value: 0.05)
    value for initial thresholding (defining clusters)
height_threshold_type: ('p-value' or 'stat', nipy default value:
    p-value)
    Is the cluster forming threshold a stat value or p-value?
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the

```

```

        interface fails to run
matlab_cmd: (a unicode string)
            matlab command to use
mfile: (a boolean, nipy default value: True)
            Run m-code using m-file
paths: (a list of items which are a directory name)
            Paths to add to matlabpath
use_fwe_correction: (a boolean, nipy default value: True)
            whether to use FWE (Bonferroni) correction for initial threshold
            (height_threshold_type has to be set to p-value)
use_mcr: (a boolean)
            Run m-code using SPM MCR
use_topo_fdr: (a boolean, nipy default value: True)
            whether to use FDR over cluster extent probabilities
use_v8struct: (a boolean, nipy default value: True)
            Generate SPM8 and higher compatible jobs

```

Outputs:

```

activation_forced: (a boolean)
cluster_forming_thr: (a float)
n_clusters: (an integer (int or long))
pre_topo_fdr_map: (an existing file name)
pre_topo_n_clusters: (an integer (int or long))
thresholded_map: (an existing file name)

```

References:: None

ThresholdStatistics[Link to code](#)

Given height and cluster size threshold calculate theoretical probabilities concerning false positives

Examples

```

>>> thresh = ThresholdStatistics()
>>> thresh.inputs.spm_mat_file = 'SPM.mat'
>>> thresh.inputs.stat_image = 'spmT_0001.img'
>>> thresh.inputs.contrast_index = 1
>>> thresh.inputs.height_threshold = 4.56
>>> thresh.run()

```

Inputs:

```

[Mandatory]
contrast_index: (an integer (int or long))
            which contrast in the SPM.mat to use
height_threshold: (a float)
            stat value for initial thresholding (defining clusters)
spm_mat_file: (an existing file name)
            absolute path to SPM.mat
stat_image: (an existing file name)
            stat image

[Optional]
extent_threshold: (an integer (int or long), nipy default value: 0)
            Minimum cluster size in voxels
ignore_exception: (a boolean, nipy default value: False)

```

```
    Print an error message instead of throwing an exception in case the
    interface fails to run
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipyne default value: True)
    Run m-code using m-file
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipyne default value: True)
    Generate SPM8 and higher compatible jobs
```

Outputs:

```
clusterwise_P_FDR: (a float)
clusterwise_P_RF: (a float)
voxelwise_P_Bonf: (a float)
voxelwise_P_FDR: (a float)
voxelwise_P_RF: (a float)
voxelwise_P_uncor: (a float)
```

References:: None

TwoSampleTTestDesign

[Link to code](#)

Create SPM design for two sample t-test

Examples

```
>>> ttest = TwoSampleTTestDesign()
>>> ttest.inputs.group1_files = ['cont1.nii', 'cont2.nii']
>>> ttest.inputs.group2_files = ['cont1a.nii', 'cont2a.nii']
>>> ttest.run()
```

Inputs:

```
[Mandatory]
group1_files: (a list of at least 2 items which are an existing file
    name)
    Group 1 input files
group2_files: (a list of at least 2 items which are an existing file
    name)
    Group 2 input files

[Optional]
covariates: (a list of items which are a dictionary with keys which
    are 'vector' or 'name' or 'interaction' or 'centering' and with
    values which are any value)
    covariate dictionary {vector, name, interaction, centering}
dependent: (a boolean)
    Are the measurements dependent between levels
explicit_mask_file: (a file name)
    use an implicit mask file to threshold
global_calc_mean: (a boolean)
    use mean for global calculation
mutually_exclusive: global_calc_omit, global_calc_values
```

```

global_calc_omit: (a boolean)
    omit global calculation
    mutually_exclusive: global_calc_mean, global_calc_values
global_calc_values: (a list of items which are a float)
    omit global calculation
    mutually_exclusive: global_calc_mean, global_calc_omit
global_normalization: (1 or 2 or 3)
    global normalization None-1, Proportional-2, ANCOVA-3
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
no_grand_mean_scaling: (a boolean)
    do not perform grand mean scaling
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
spm_mat_dir: (an existing directory name)
    directory to store SPM.mat file (opt)
threshold_mask_absolute: (a float)
    use an absolute threshold
    mutually_exclusive: threshold_mask_none, threshold_mask_relative
threshold_mask_none: (a boolean)
    do not use threshold masking
    mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
threshold_mask_relative: (a float)
    threshold using a proportion of the global value
    mutually_exclusive: threshold_mask_absolute, threshold_mask_none
unequal_variance: (a boolean)
    Are the variances equal or unequal between groups
use_implicit_threshold: (a boolean)
    use implicit mask NaNs or zeros to threshold
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs

```

Outputs:

```

spm_mat_file: (an existing file name)
    SPM mat file

```

References:: None

interfaces.spm.preprocess

ApplyDeformations

[Link to code](#)

Inputs:

```

[Mandatory]
deformation_field: (an existing file name)
in_files: (a list of items which are an existing file name)
reference_volume: (an existing file name)

```

```
[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interp: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
```

Outputs:

```
out_files: (a list of items which are an existing file name)
```

References:: None

Coregister

[Link to code](#)

Use `spm_coreg` for estimating cross-modality rigid body alignment

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=39>

Examples

```
>>> import nipy.interfaces.spm as spm
>>> coreg = spm.Coregister()
>>> coreg.inputs.target = 'functional.nii'
>>> coreg.inputs.source = 'structural.nii'
>>> coreg.run()
```

Inputs:

```
[Mandatory]
source: (a list of items which are an existing file name)
    file to register to target
target: (an existing file name)
    reference file to register to

[Optional]
apply_to_files: (a list of items which are an existing file name)
    files to apply transformation to
cost_function: ('mi' or 'nmi' or 'ecc' or 'ncc')
    cost function, one of:
    'mi' - Mutual Information,
    'nmi' - Normalised Mutual Information,
    'ecc' - Entropy Correlation Coefficient,
    'ncc' - Normalised Cross Correlation
fwhm: (a list of from 2 to 2 items which are a float)
    gaussian smoothing kernel width (mm)
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
```

```

        interface fails to run
    jobtype: ('estwrite' or 'estimate' or 'write', nipy default value:
        estwrite)
        one of: estimate, write, estwrite
    matlab_cmd: (a unicode string)
        matlab command to use
    mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
    out_prefix: (a string, nipy default value: r)
        coregistered output prefix
    paths: (a list of items which are a directory name)
        Paths to add to matlabpath
    separation: (a list of items which are a float)
        sampling separation in mm
    tolerance: (a list of items which are a float)
        acceptable tolerance for each of 12 params
    use_mcr: (a boolean)
        Run m-code using SPM MCR
    use_v8struct: (a boolean, nipy default value: True)
        Generate SPM8 and higher compatible jobs
    write_interp: (0 <= a long integer <= 7)
        degree of b-spline used for interpolation
    write_mask: (a boolean)
        True/False mask output image
    write_wrap: (a list of from 3 to 3 items which are an integer (int or
        long))
        Check if interpolation should wrap in [x,y,z]

```

Outputs:

```

coregistered_files: (a list of items which are an existing file name)
    Coregistered other files
coregistered_source: (a list of items which are an existing file
    name)
    Coregistered source files

```

References:: None

CreateWarped[Link to code](#)

Apply a flow field estimated by DARTEL to create warped images

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=190>**Examples**

```

>>> import nipy.interfaces.spm as spm
>>> create_warped = spm.CreateWarped()
>>> create_warped.inputs.image_files = ['rcls1.nii', 'rcls2.nii']
>>> create_warped.inputs.flowfield_files = ['u_rcls1_Template.nii', 'u_rcls2_
↳Template.nii']
>>> create_warped.run()

```

Inputs:

```

[Mandatory]
flowfield_files: (a list of items which are an existing file name)
    DARTEL flow fields u_rcl*

```

```

image_files: (a list of items which are an existing file name)
    A list of files to be warped

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interp: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
iterations: (0 <= a long integer <= 9)
    The number of iterations: log2(number of time steps)
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
modulate: (a boolean)
    Modulate images
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs

```

Outputs:

```
warped_files: (a list of items which are an existing file name)
```

References:: None

DARTEL

Link to code

Use spm DARTEL to create a template and flow fields

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=185>**Examples**

```

>>> import nipy.interfaces.spm as spm
>>> dartel = spm.DARTEL()
>>> dartel.inputs.image_files = [['rc1s1.nii', 'rc1s2.nii'], ['rc2s1.nii', 'rc2s2.
↪nii']]
>>> dartel.run()

```

Inputs:

```

[Mandatory]
image_files: (a list of items which are a list of items which are an
    existing file name)
    A list of files to be segmented

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
iteration_parameters: (a list of from 3 to 12 items which are a tuple
    of the form: (1 <= a long integer <= 10, a tuple of the form: (a
    float, a float, a float), 1 or 2 or 4 or 8 or 16 or 32 or 64 or 128

```



```

    or 256 or 512, 0 or 0.5 or 1 or 2 or 4 or 8 or 16 or 32))
List of tuples for each iteration
- Inner iterations
- Regularization parameters
- Time points for deformation model
- smoothing parameter
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
optimization_parameters: (a tuple of the form: (a float, 1 <= a long
    integer <= 8, 1 <= a long integer <= 8))
    Optimization settings a tuple
    - LM regularization
    - cycles of multigrid solver
    - relaxation iterations
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
regularization_form: ('Linear' or 'Membrane' or 'Bending')
    Form of regularization energy term
template_prefix: (a unicode string, nipy default value: Template)
    Prefix for template
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs

```

Outputs:

```

dartel_flow_fields: (a list of items which are an existing file name)
    DARTEL flow fields
final_template_file: (an existing file name)
    final DARTEL template
template_files: (a list of items which are an existing file name)
    Templates from different stages of iteration

```

References:: None

DARTElNorm2MNI

Link to code

Use spm DARTEL to normalize data to MNI space

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=188>**Examples**

```

>>> import nipy.interfaces.spm as spm
>>> nm = spm.DARTElNorm2MNI()
>>> nm.inputs.template_file = 'Template_6.nii'
>>> nm.inputs.flowfield_files = ['u_rcls1_Template.nii', 'u_rcls3_Template.nii']
>>> nm.inputs.apply_to_files = ['cls1.nii', 'cls3.nii']
>>> nm.inputs.modulate = True
>>> nm.run()

```

Inputs:

```

[Mandatory]
apply_to_files: (a list of items which are an existing file name)

```

```

Files to apply the transform to
flowfield_files: (a list of items which are an existing file name)
DARTEL flow fields u_rcl*
template_file: (an existing file name)
DARTEL template

[Optional]
bounding_box: (a tuple of the form: (a float, a float, a float, a
float, a float, a float))
Voxel sizes for output file
fwhm: (a list of from 3 to 3 items which are a float or a float)
3-list of fwhm for each dimension
ignore_exception: (a boolean, nipy default value: False)
Print an error message instead of throwing an exception in case the
interface fails to run
matlab_cmd: (a unicode string)
matlab command to use
mfile: (a boolean, nipy default value: True)
Run m-code using m-file
modulate: (a boolean)
Modulate out images - no modulation preserves concentrations
paths: (a list of items which are a directory name)
Paths to add to matlabpath
use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
Generate SPM8 and higher compatible jobs
voxel_size: (a tuple of the form: (a float, a float, a float))
Voxel sizes for output file

```

Outputs:

```

normalization_parameter_file: (an existing file name)
Transform parameters to MNI space
normalized_files: (a list of items which are an existing file name)
Normalized files in MNI space

```

References:: None

NewSegment[Link to code](#)

Use `spm_preproc8` (New Segment) to separate structural images into different tissue classes. Supports multiple modalities.

NOTE: This interface currently supports single channel input only

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=43>

Examples

```

>>> import nipy.interfaces.spm as spm
>>> seg = spm.NewSegment()
>>> seg.inputs.channel_files = 'structural.nii'
>>> seg.inputs.channel_info = (0.0001, 60, (True, True))
>>> seg.run()

```

For VBM pre-processing [<http://www.fil.ion.ucl.ac.uk/~john/misc/VBMclass10.pdf>], TPM.nii should be replaced by /path/to/spm8/toolbox/Seg/TPM.nii

```

>>> seg = NewSegment()
>>> seg.inputs.channel_files = 'structural.nii'
>>> tissue1 = (('TPM.nii', 1), 2, (True,True), (False, False))
>>> tissue2 = (('TPM.nii', 2), 2, (True,True), (False, False))
>>> tissue3 = (('TPM.nii', 3), 2, (True,False), (False, False))
>>> tissue4 = (('TPM.nii', 4), 2, (False,False), (False, False))
>>> tissue5 = (('TPM.nii', 5), 2, (False,False), (False, False))
>>> seg.inputs.tissues = [tissue1, tissue2, tissue3, tissue4, tissue5]
>>> seg.run()

```

Inputs:

```

[Mandatory]
channel_files: (a list of items which are an existing file name)
    A list of files to be segmented

[Optional]
affine_regularization: ('mni' or 'eastern' or 'subj' or 'none')
    mni, eastern, subj, none
channel_info: (a tuple of the form: (a float, a float, a tuple of the
    form: (a boolean, a boolean)))
    A tuple with the following fields:
    - bias regularisation (0-10)
    - FWHM of Gaussian smoothness of bias
    - which maps to save (Corrected, Field) - a tuple of two boolean
      values
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
sampling_distance: (a float)
    Sampling distance on data for parameter estimation
tissues: (a list of items which are a tuple of the form: (a tuple of
    the form: (an existing file name, an integer (int or long)), an
    integer (int or long), a tuple of the form: (a boolean, a boolean),
    a tuple of the form: (a boolean, a boolean)))
    A list of tuples (one per tissue) with the following fields:
    - tissue probability map (4D), 1-based index to frame
    - number of gaussians
    - which maps to save [Native, DARTEL] - a tuple of two boolean
      values
    - which maps to save [Unmodulated, Modulated] - a tuple of two
      boolean values
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPM8 and higher compatible jobs
warping_regularization: (a list of from 5 to 5 items which are a
    float or a float)
    Warping regularization parameter(s). Accepts float or list of floats
    (the latter is required by SPM12)
write_deformation_fields: (a list of from 2 to 2 items which are a
    boolean)

```

Which deformation fields to write:[Inverse, Forward]

Outputs:

```

bias_corrected_images: (a list of items which are an existing file
                        name)
                        bias corrected images
bias_field_images: (a list of items which are an existing file name)
                    bias field images
dartel_input_images: (a list of items which are a list of items which
                      are an existing file name)
                      dartel imported class images
forward_deformation_field: (a list of items which are an existing
                            file name)
inverse_deformation_field: (a list of items which are an existing
                             file name)
modulated_class_images: (a list of items which are a list of items
                         which are an existing file name)
                         modulated+normalized class images
native_class_images: (a list of items which are a list of items which
                      are an existing file name)
                      native space probability maps
normalized_class_images: (a list of items which are a list of items
                          which are an existing file name)
                          normalized class images
transformation_mat: (a list of items which are an existing file name)
                    Normalization transformation

```

References:: None

Normalize

[Link to code](#)

use `spm_normalise` for warping an image to a template

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=203>

Examples

```

>>> import nipy.interfaces.spm as spm
>>> norm = spm.Normalize()
>>> norm.inputs.source = 'functional.nii'
>>> norm.run()

```

Inputs:

```

[Mandatory]
parameter_file: (a file name)
                 normalization parameter file*_sn.mat
                 mutually_exclusive: source, template
source: (a list of items which are an existing file name)
        file to normalize to template
        mutually_exclusive: parameter_file
template: (an existing file name)
          template file to normalize to
          mutually_exclusive: parameter_file

[Optional]
DCT_period_cutoff: (a float)

```

```

    Cutoff of for DCT bases
affine_regularization_type: ('mni' or 'size' or 'none')
    mni, size, none
apply_to_files: (a list of items which are an existing file name or a
    list of items which are an existing file name)
    files to apply transformation to
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
jobtype: ('estwrite' or 'est' or 'write', nipype default value:
    estwrite)
    Estimate, Write or do both
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipype default value: True)
    Run m-code using m-file
nonlinear_iterations: (an integer (int or long))
    Number of iterations of nonlinear warping
nonlinear_regularization: (a float)
    the amount of the regularization for the nonlinear part of the
    normalization
out_prefix: (a string, nipype default value: w)
    normalized output prefix
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
source_image_smoothing: (a float)
    source smoothing
source_weight: (a file name)
    name of weighting image for source
template_image_smoothing: (a float)
    template smoothing
template_weight: (a file name)
    name of weighting image for template
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
    Generate SPM8 and higher compatible jobs
write_bounding_box: (a list of from 2 to 2 items which are a list of
    from 3 to 3 items which are a float)
    3x2-element list of lists
write_interp: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
write_preserve: (a boolean)
    True/False warped images are modulated
write_voxel_sizes: (a list of from 3 to 3 items which are a float)
    3-element list
write_wrap: (a list of items which are an integer (int or long))
    Check if interpolation should wrap in [x,y,z] - list of bools

```

Outputs:

```

normalization_parameters: (a list of items which are an existing file
    name)
    MAT files containing the normalization parameters
normalized_files: (a list of items which are an existing file name)
    Normalized other files
normalized_source: (a list of items which are an existing file name)
    Normalized source files

```

References:: None

Normalize12

[Link to code](#)

uses SPM12's new Normalise routine for warping an image to a template. Spatial normalisation is now done via the segmentation routine (which was known as New Segment in SPM8). Note that the normalisation in SPM12 is done towards a file containing multiple tissue probability maps, which was not the case in SPM8. <http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=49>

Examples

```
>>> import nipy.interfaces.spm as spm
>>> norm12 = spm.Normalize12()
>>> norm12.inputs.image_to_align = 'structural.nii'
>>> norm12.inputs.apply_to_files = 'functional.nii'
>>> norm12.run()
```

Inputs:

```
[Mandatory]
deformation_file: (a file name)
    file y_*.nii containing 3 deformation fields for the deformation in
    x, y and z dimension
    mutually_exclusive: image_to_align, tpm
image_to_align: (an existing file name)
    file to estimate normalization parameters with
    mutually_exclusive: deformation_file

[Optional]
affine_regularization_type: ('mni' or 'size' or 'none')
    mni, size, none
apply_to_files: (a list of items which are an existing file name or a
    list of items which are an existing file name)
    files to apply transformation to
bias_fwhm: (30 or 40 or 50 or 60 or 70 or 80 or 90 or 100 or 110 or
    120 or 130 or 140 or 150 or 'Inf')
    FWHM of Gaussian smoothness of bias
bias_regularization: (0 or 1e-05 or 0.0001 or 0.001 or 0.01 or 0.1 or
    1 or 10)
    no(0) - extremely heavy (10)
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
jobtype: ('estwrite' or 'est' or 'write', nipy default value:
    estwrite)
    Estimate, Write or do Both
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
out_prefix: (a string, nipy default value: w)
    Normalized output prefix
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
sampling_distance: (a float)
    Sampling distance on data for parameter estimation
smoothness: (a float)
```

```

    value (in mm) to smooth the data before normalization
tpm: (an existing file name)
    template in form of tissue probablitiy maps to normalize to
    mutually_exclusive: deformation_file
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
warping_regularization: (a list of from 5 to 5 items which are a
    float)
    controls balance between parameters and data
write_bounding_box: (a list of from 2 to 2 items which are a list of
    from 3 to 3 items which are a float)
    3x2-element list of lists representing the bounding box (in mm) to
    be written
write_interp: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
write_voxel_sizes: (a list of from 3 to 3 items which are a float)
    3-element list representing the voxel sizes (in mm) of the written
    normalised images

```

Outputs:

```

deformation_field: (a list of items which are an existing file name)
    NIfTI file containing 3 deformation fields for the deformation in x,
    y and z dimension
normalized_files: (a list of items which are an existing file name)
    Normalized other files
normalized_image: (a list of items which are an existing file name)
    Normalized file that needed to be aligned

```

References:: None

Realign[Link to code](#)Use `spm_realign` for estimating within modality rigid body alignment<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=25>**Examples**

```

>>> import nipy.interfaces.spm as spm
>>> realign = spm.Realign()
>>> realign.inputs.in_files = 'functional.nii'
>>> realign.inputs.register_to_mean = True
>>> realign.run()

```

Inputs:

```

[Mandatory]
in_files: (a list of items which are a list of items which are an
    existing file name or an existing file name)
    list of filenames to realign

[Optional]
fwhm: (a floating point number >= 0.0)
    gaussian smoothing kernel width
ignore_exception: (a boolean, nipy default value: False)

```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
interp: (0 <= a long integer <= 7)
        degree of b-spline used for interpolation
jobtype: ('estwrite' or 'estimate' or 'write', nipy default value:
        estwrite)
        one of: estimate, write, estwrite
matlab_cmd: (a unicode string)
        matlab command to use
mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
out_prefix: (a string, nipy default value: r)
        realigned output prefix
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
quality: (0.0 <= a floating point number <= 1.0)
        0.1 = fast, 1.0 = precise
register_to_mean: (a boolean)
        Indicate whether realignment is done to the mean image
separation: (a floating point number >= 0.0)
        sampling separation in mm
use_mcr: (a boolean)
        Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
        Generate SPM8 and higher compatible jobs
weight_img: (an existing file name)
        filename of weighting image
wrap: (a list of from 3 to 3 items which are an integer (int or
        long))
        Check if interpolation should wrap in [x,y,z]
write_interp: (0 <= a long integer <= 7)
        degree of b-spline used for interpolation
write_mask: (a boolean)
        True/False mask output image
write_which: (a list of items which are a value of class 'int',
        nipy default value: [2, 1])
        determines which images to reslice
write_wrap: (a list of from 3 to 3 items which are an integer (int or
        long))
        Check if interpolation should wrap in [x,y,z]

```

Outputs:

```

mean_image: (an existing file name)
        Mean image file from the realignment
modified_in_files: (a list of items which are a list of items which
        are an existing file name or an existing file name)
        Copies of all files passed to in_files. Headers will have been
        modified to align all images with the first, or optionally to first
        do that, extract a mean image, and re-align to that mean image.
realigned_files: (a list of items which are a list of items which are
        an existing file name or an existing file name)
        If jobtype is write or estwrite, these will be the resliced files.
        Otherwise, they will be copies of in_files that have had their
        headers rewritten.
realignment_parameters: (a list of items which are an existing file
        name)
        Estimated translation and rotation parameters

```


References:: None

Segment

[Link to code](#)

use `spm_segment` to separate structural images into different tissue classes.

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=209>

Examples

```
>>> import nipy.interfaces.spm as spm
>>> seg = spm.Segment()
>>> seg.inputs.data = 'structural.nii'
>>> seg.run()
```

Inputs:

```
[Mandatory]
data: (a list of items which are an existing file name)
      one scan per subject

[Optional]
affine_regularization: ('mni' or 'eastern' or 'subj' or 'none' or '')
      Possible options: "mni", "eastern", "subj", "none" (no
      regularisation), "" (no affine registration)
bias_fwhm: (30 or 40 or 50 or 60 or 70 or 80 or 90 or 100 or 110 or
            120 or 130 or 'Inf')
      FWHM of Gaussian smoothness of bias
bias_regularization: (0 or 1e-05 or 0.0001 or 0.001 or 0.01 or 0.1 or
                     1 or 10)
      no(0) - extremely heavy (10)
clean_masks: ('no' or 'light' or 'thorough')
      clean using estimated brain mask ('no', 'light', 'thorough')
csf_output_type: (a list of from 3 to 3 items which are a boolean)
      Options to produce CSF images: c3*.img, wc3*.img and mwc3*.img.
      None: [False, False, False],
      Native Space: [False, False, True],
      Unmodulated Normalised: [False, True, False],
      Modulated Normalised: [True, False, False],
      Native + Unmodulated Normalised: [False, True, True],
      Native + Modulated Normalised: [True, False, True],
      Native + Modulated + Unmodulated: [True, True, True],
      Modulated + Unmodulated Normalised: [True, True, False]
gaussians_per_class: (a list of items which are an integer (int or
                      long))
      num Gaussians capture intensity distribution
gm_output_type: (a list of from 3 to 3 items which are a boolean)
      Options to produce grey matter images: c1*.img, wc1*.img and
      mwc1*.img.
      None: [False, False, False],
      Native Space: [False, False, True],
      Unmodulated Normalised: [False, True, False],
      Modulated Normalised: [True, False, False],
      Native + Unmodulated Normalised: [False, True, True],
      Native + Modulated Normalised: [True, False, True],
      Native + Modulated + Unmodulated: [True, True, True],
      Modulated + Unmodulated Normalised: [True, True, False]
ignore_exception: (a boolean, nipy default value: False)
```

```

        Print an error message instead of throwing an exception in case the
        interface fails to run
mask_image: (an existing file name)
        Binary image to restrict parameter estimation
matlab_cmd: (a unicode string)
        matlab command to use
mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
sampling_distance: (a float)
        Sampling distance on data for parameter estimation
save_bias_corrected: (a boolean)
        True/False produce a bias corrected image
tissue_prob_maps: (a list of items which are an existing file name)
        list of gray, white & csf prob. (opt,)
use_mcr: (a boolean)
        Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
        Generate SPM8 and higher compatible jobs
warp_frequency_cutoff: (a float)
        Cutoff of DCT bases
warping_regularization: (a float)
        Controls balance between parameters and data
wm_output_type: (a list of from 3 to 3 items which are a boolean)
        Options to produce white matter images: c2*.img, wc2*.img and
        mwc2*.img.
        None: [False, False, False],
        Native Space: [False, False, True],
        Unmodulated Normalised: [False, True, False],
        Modulated Normalised: [True, False, False],
        Native + Unmodulated Normalised: [False, True, True],
        Native + Modulated Normalised: [True, False, True],
        Native + Modulated + Unmodulated: [True, True, True],
        Modulated + Unmodulated Normalised: [True, True, False]

```

Outputs:

```

bias_corrected_image: (a file name)
        bias-corrected version of input image
inverse_transformation_mat: (an existing file name)
        Inverse normalization info
modulated_csf_image: (a file name)
        modulated, normalized csf probability map
modulated_gm_image: (a file name)
        modulated, normalized grey probability map
modulated_input_image: (a file name)
        bias-corrected version of input image
modulated_wm_image: (a file name)
        modulated, normalized white probability map
native_csf_image: (a file name)
        native space csf probability map
native_gm_image: (a file name)
        native space grey probability map
native_wm_image: (a file name)
        native space white probability map
normalized_csf_image: (a file name)
        normalized csf probability map

```

```

normalized_gm_image: (a file name)
    normalized grey probability map
normalized_wm_image: (a file name)
    normalized white probability map
transformation_mat: (an existing file name)
    Normalization transformation

```

References:: None

SliceTiming

[Link to code](#)

Use `spm` to perform slice timing correction.

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=19>

Examples

```

>>> from nipy.interfaces.spm import SliceTiming
>>> st = SliceTiming()
>>> st.inputs.in_files = 'functional.nii'
>>> st.inputs.num_slices = 32
>>> st.inputs.time_repetition = 6.0
>>> st.inputs.time_acquisition = 6. - 6./32.
>>> st.inputs.slice_order = list(range(32,0,-1))
>>> st.inputs.ref_slice = 1
>>> st.run()

```

Inputs:

```

[Mandatory]
in_files: (a list of items which are a list of items which are an
    existing file name or an existing file name)
    list of filenames to apply slice timing
num_slices: (an integer (int or long))
    number of slices in a volume
ref_slice: (an integer (int or long))
    1-based Number of the reference slice or reference time point if
    slice_order is in onsets (ms)
slice_order: (a list of items which are a float)
    1-based order or onset (in ms) in which slices are acquired
time_acquisition: (a float)
    time of volume acquisition. usuallycalculated as TR-(TR/num_slices)
time_repetition: (a float)
    time between volume acquisitions(start to start time)

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
out_prefix: (a string, nipy default value: a)
    slicetimed output prefix
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
use_mcr: (a boolean)

```

```
Run m-code using SPM MCR
use_v8struct: (a boolean, nipyre default value: True)
Generate SPM8 and higher compatible jobs
```

Outputs:

```
timecorrected_files: (a list of items which are a list of items which
are an existing file name or an existing file name)
slice time corrected files
```

References:: None

Smooth

[Link to code](#)Use `spm_smooth` for 3D Gaussian smoothing of image volumes.<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=55>

Examples

```
>>> import nipyre.interfaces.spm as spm
>>> smooth = spm.Smooth()
>>> smooth.inputs.in_files = 'functional.nii'
>>> smooth.inputs.fwhm = [4, 4, 4]
>>> smooth.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
list of files to smooth

[Optional]
data_type: (an integer (int or long))
Data type of the output images
fwhm: (a list of from 3 to 3 items which are a float or a float)
3-list of fwhm for each dimension
ignore_exception: (a boolean, nipyre default value: False)
Print an error message instead of throwing an exception in case the
interface fails to run
implicit_masking: (a boolean)
A mask implied by a particular voxel value
matlab_cmd: (a unicode string)
matlab command to use
mfile: (a boolean, nipyre default value: True)
Run m-code using m-file
out_prefix: (a string, nipyre default value: s)
smoothed output prefix
paths: (a list of items which are a directory name)
Paths to add to matlabpath
use_mcr: (a boolean)
Run m-code using SPM MCR
use_v8struct: (a boolean, nipyre default value: True)
Generate SPM8 and higher compatible jobs
```

Outputs:

```
smoothed_files: (a list of items which are an existing file name)
smoothed files
```

References:: None

VBMSegment

[Link to code](#)

Use VBM8 toolbox to separate structural images into different tissue classes.

Example

```
>>> import nipy.interfaces.spm as spm
>>> seg = spm.VBMSegment()
>>> seg.inputs.tissues = 'TPM.nii'
>>> seg.inputs.dartel_template = 'Template_1_IXI550_MNI152.nii'
>>> seg.inputs.bias_corrected_native = True
>>> seg.inputs.gm_native = True
>>> seg.inputs.wm_native = True
>>> seg.inputs.csf_native = True
>>> seg.inputs.pve_label_native = True
>>> seg.inputs.deformation_field = (True, False)
>>> seg.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
          A list of files to be segmented

[Optional]
bias_corrected_affine: (a boolean, nipy default value: False)
bias_corrected_native: (a boolean, nipy default value: False)
bias_corrected_normalized: (a boolean, nipy default value: True)
bias_fwhm: (30 or 40 or 50 or 60 or 70 or 80 or 90 or 100 or 110 or
            120 or 130 or 'Inf', nipy default value: 60)
            FWHM of Gaussian smoothness of bias
bias_regularization: (0 or 1e-05 or 0.0001 or 0.001 or 0.01 or 0.1 or
                     1 or 10, nipy default value: 0.0001)
                     no(0) - extremely heavy (10)
cleanup_partitions: (an integer (int or long), nipy default value:
                    1)
                    0=None,1=light,2=thorough
csf_dartel: (0 <= a long integer <= 2, nipy default value: 0)
            0=None,1=rigid(SPM8 default),2=affine
csf_modulated_normalized: (0 <= a long integer <= 2, nipy default
                           value: 2)
                           0=none,1=affine+non-linear(SPM8 default),2=non-linear only
csf_native: (a boolean, nipy default value: False)
csf_normalized: (a boolean, nipy default value: False)
dartel_template: (an existing file name)
deformation_field: (a tuple of the form: (a boolean, a boolean),
                   nipy default value: (0, 0))
                   forward and inverse field
display_results: (a boolean, nipy default value: True)
gaussians_per_class: (a tuple of the form: (an integer (int or long),
      an integer (int or long), an integer (int or long), an integer (int
      or long), an integer (int or long), an integer (int or long)),
      nipy default value: (2, 2, 2, 3, 4, 2))
      number of gaussians for each tissue class
gm_dartel: (0 <= a long integer <= 2, nipy default value: 0)
```

```

    0=None,1=rigid(SPM8 default),2=affine
gm_modulated_normalized: (0 <= a long integer <= 2, nipy default
    value: 2)
    0=none,1=affine+non-linear(SPM8 default),2=non-linear only
gm_native: (a boolean, nipy default value: False)
gm_normalized: (a boolean, nipy default value: False)
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
jacobian_determinant: (a boolean, nipy default value: False)
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
mrf_weighting: (a float, nipy default value: 0.15)
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
pve_label_dartel: (0 <= a long integer <= 2, nipy default value: 0)
    0=None,1=rigid(SPM8 default),2=affine
pve_label_native: (a boolean, nipy default value: False)
pve_label_normalized: (a boolean, nipy default value: False)
sampling_distance: (a float, nipy default value: 3)
    Sampling distance on data for parameter estimation
spatial_normalization: ('high' or 'low', nipy default value: high)
tissues: (an existing file name)
    tissue probability map
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_sanlm_denoising_filter: (0 <= a long integer <= 2, nipy default
    value: 2)
    0=No denoising, 1=denoising,2=denoising multi-threaded
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
warping_regularization: (a float, nipy default value: 4)
    Controls balance between parameters and data
wm_dartel: (0 <= a long integer <= 2, nipy default value: 0)
    0=None,1=rigid(SPM8 default),2=affine
wm_modulated_normalized: (0 <= a long integer <= 2, nipy default
    value: 2)
    0=none,1=affine+non-linear(SPM8 default),2=non-linear only
wm_native: (a boolean, nipy default value: False)
wm_normalized: (a boolean, nipy default value: False)

```

Outputs:

```

bias_corrected_images: (a list of items which are an existing file
    name)
    bias corrected images
dartel_input_images: (a list of items which are a list of items which
    are an existing file name)
    dartel imported class images
forward_deformation_field: (a list of items which are an existing
    file name)
inverse_deformation_field: (a list of items which are an existing
    file name)
jacobian_determinant_images: (a list of items which are an existing
    file name)
modulated_class_images: (a list of items which are a list of items

```

```

        which are an existing file name)
        modulated+normalized class images
native_class_images: (a list of items which are a list of items which
        are an existing file name)
        native space probability maps
normalized_bias_corrected_images: (a list of items which are an
        existing file name)
        bias corrected images
normalized_class_images: (a list of items which are a list of items
        which are an existing file name)
        normalized class images
pve_label_native_images: (a list of items which are an existing file
        name)
pve_label_normalized_images: (a list of items which are an existing
        file name)
pve_label_registered_images: (a list of items which are an existing
        file name)
transformation_mat: (a list of items which are an existing file name)
        Normalization transformation

```

References:: None

interfaces.spm.utils

Analyze2nii

[Link to code](#)

Inputs:

```

[Mandatory]
analyze_file: (an existing file name)

[Optional]
ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
matlab_cmd: (a unicode string)
        matlab command to use
mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
use_mcr: (a boolean)
        Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
        Generate SPM8 and higher compatible jobs

```

Outputs:

```

ignore_exception: (a boolean, nipy default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
matlab_cmd: (a unicode string)
        matlab command to use
mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
nifti_file: (an existing file name)
paths: (a list of items which are a directory name)

```

```
    Paths to add to matlabpath
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipyne default value: True)
    Generate SPM8 and higher compatible jobs
```

References:: None

ApplyInverseDeformation

[Link to code](#)

Uses spm to apply inverse deformation stored in a .mat file or a deformation field to a given file

Examples

```
>>> import nipyne.interfaces.spm.utils as spmu
>>> inv = spmu.ApplyInverseDeformation()
>>> inv.inputs.in_files = 'functional.nii'
>>> inv.inputs.deformation = 'struct_to_func.mat'
>>> inv.inputs.target = 'structural.nii'
>>> inv.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
    Files on which deformation is applied

[Optional]
bounding_box: (a list of from 6 to 6 items which are a float)
    6-element list (opt)
deformation: (an existing file name)
    SN SPM deformation file
    mutually_exclusive: deformation_field
deformation_field: (an existing file name)
    SN SPM deformation file
    mutually_exclusive: deformation
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interpolation: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipyne default value: True)
    Run m-code using m-file
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
target: (an existing file name)
    File defining target space
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipyne default value: True)
    Generate SPM8 and higher compatible jobs
voxel_sizes: (a list of from 3 to 3 items which are a float)
    3-element list (opt)
```

Outputs:


```
out_files: (a list of items which are an existing file name)
           Transformed files
```

References:: None

ApplyTransform

[Link to code](#)

Uses SPM to apply transform stored in a .mat file to given file

Examples

```
>>> import nipy.interfaces.spm.utils as spmu
>>> applymat = spmu.ApplyTransform()
>>> applymat.inputs.in_file = 'functional.nii'
>>> applymat.inputs.mat = 'func_to_struct.mat'
>>> applymat.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         file to apply transform to, (only updates header)
mat: (an existing file name)
     file holding transform to apply

[Optional]
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
matlab_cmd: (a unicode string)
            matlab command to use
mfile: (a boolean, nipy default value: True)
       Run m-code using m-file
out_file: (a file name)
          output file name for transformed data
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
use_mcr: (a boolean)
         Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
              Generate SPM8 and higher compatible jobs
```

Outputs:

```
out_file: (an existing file name)
          Transformed image file
```

References:: None

CalcCoregAffine

[Link to code](#)

Uses SPM (spm_coreg) to calculate the transform mapping moving to target. Saves Transform in mat (matlab binary file) Also saves inverse transform

Examples

```
>>> import nipyne.interfaces.spm.utils as spmu
>>> coreg = spmu.CalcCoregAffine(matlab_cmd='matlab-spm8')
>>> coreg.inputs.target = 'structural.nii'
>>> coreg.inputs.moving = 'functional.nii'
>>> coreg.inputs.mat = 'func_to_struct.mat'
>>> coreg.run()
```

Note:

- the output file mat is saved as a matlab binary file
 - calculating the transforms does NOT change either input image it does not **move** the moving image, only calculates the transform that can be used to move it
-

Inputs:

```
[Mandatory]
moving: (an existing file name)
        volume transform can be applied to register with target
target: (an existing file name)
        target for generating affine transform

[Optional]
ignore_exception: (a boolean, nipyne default value: False)
        Print an error message instead of throwing an exception in case the
        interface fails to run
invmat: (a file name)
        Filename used to store inverse affine matrix
mat: (a file name)
        Filename used to store affine matrix
matlab_cmd: (a unicode string)
        matlab command to use
mfile: (a boolean, nipyne default value: True)
        Run m-code using m-file
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
use_mcr: (a boolean)
        Run m-code using SPM MCR
use_v8struct: (a boolean, nipyne default value: True)
        Generate SPM8 and higher compatible jobs
```

Outputs:

```
invmat: (a file name)
        Matlab file holding inverse transform
mat: (an existing file name)
        Matlab file holding transform
```

References:: None

DicomImport

[Link to code](#)

Uses spm to convert DICOM files to nii or img+hdr.

Examples

```
>>> import nipy.interfaces.spm.utils as spmu
>>> di = spmu.DicomImport()
>>> di.inputs.in_files = ['functional_1.dcm', 'functional_2.dcm']
>>> di.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
          dicom files to be converted

[Optional]
format: ('nii' or 'img', nipy default value: nii)
        output format.
icedims: (a boolean, nipy default value: False)
          If image sorting fails, one can try using the additional SIEMENS
          ICEDims information to create unique filenames. Use this only if
          there would be multiple volumes with exactly the same file names.
ignore_exception: (a boolean, nipy default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
matlab_cmd: (a unicode string)
             matlab command to use
mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
output_dir: (a unicode string, nipy default value:
             ./converted_dicom)
             output directory.
output_dir_struct: ('flat' or 'series' or 'patname' or 'patid_date'
                   or 'patid' or 'date_time', nipy default value: flat)
                   directory structure for the output.
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
use_mcr: (a boolean)
          Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
               Generate SPM8 and higher compatible jobs
```

Outputs:

```
out_files: (a list of items which are an existing file name)
            converted files
```

References:: None

Reslice

[Link to code](#)

uses `spm_reslice` to resample `in_file` into space of `space_defining`

Inputs:

```
[Mandatory]
in_file: (an existing file name)
          file to apply transform to, (only updates header)
space_defining: (an existing file name)
                 Volume defining space to slice in_file into
```

```
[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interp: (0 <= a long integer <= 7, nipy default value: 0)
    degree of b-spline used for interpolation0 is nearest neighbor
    (default)
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
out_file: (a file name)
    Optional file to save resliced volume
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
```

Outputs:

```
out_file: (an existing file name)
    resliced volume
```

References:: None

ResliceToReference[Link to code](#)

Uses spm to reslice a volume to a target image space or to a provided voxel size and bounding box

Examples

```
>>> import nipy.interfaces.spm.utils as spmu
>>> r2ref = spmu.ResliceToReference()
>>> r2ref.inputs.in_files = 'functional.nii'
>>> r2ref.inputs.target = 'structural.nii'
>>> r2ref.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
    Files on which deformation is applied

[Optional]
bounding_box: (a list of from 6 to 6 items which are a float)
    6-element list (opt)
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
interpolation: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
matlab_cmd: (a unicode string)
    matlab command to use
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
```

```
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
target: (an existing file name)
        File defining target space
use_mcr: (a boolean)
        Run m-code using SPM MCR
use_v8struct: (a boolean, nipyype default value: True)
        Generate SPM8 and higher compatible jobs
voxel_sizes: (a list of from 3 to 3 items which are a float)
        3-element list (opt)
```

Outputs:

```
out_files: (a list of items which are an existing file name)
        Transformed files
```

References:: None

interfaces.utility.base

AssertEqual

[Link to code](#)

Inputs:

```
[Mandatory]
volume1: (an existing file name)
volume2: (an existing file name)

[Optional]
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

Outputs:

```
None
```

IdentityInterface

[Link to code](#)

Basic interface class generates identity mappings

Examples

```
>>> from nipype.interfaces.utility import IdentityInterface
>>> ii = IdentityInterface(fields=['a', 'b'], mandatory_inputs=False)
>>> ii.inputs.a
<undefined>
```

```
>>> ii.inputs.a = 'foo'
>>> out = ii._outputs()
>>> out.a
<undefined>
```

```
>>> out = ii.run()
>>> out.outputs.a
'foo'
```

```
>>> ii2 = IdentityInterface(fields=['a', 'b'], mandatory_inputs=True)
>>> ii2.inputs.a = 'foo'
>>> out = ii2.run()
ValueError: IdentityInterface requires a value for input 'b' because it was_
↳listed in 'fields' Interface IdentityInterface failed to run.
```

Inputs:

```
None
```

Outputs:

```
None
```

Merge

[Link to code](#)

Basic interface class to merge inputs into a single list

Merge(1) will merge a list of lists

Examples

```
>>> from nipyne.interfaces.utility import Merge
>>> mi = Merge(3)
>>> mi.inputs.in1 = 1
>>> mi.inputs.in2 = [2, 5]
>>> mi.inputs.in3 = 3
>>> out = mi.run()
>>> out.outputs.out
[1, 2, 5, 3]
```

```
>>> merge = Merge(1)
>>> merge.inputs.in1 = [1, [2, 5], 3]
>>> out = merge.run()
>>> out.outputs.out
[1, [2, 5], 3]
```

```
>>> merge = Merge(1)
>>> merge.inputs.in1 = [1, [2, 5], 3]
>>> merge.inputs.ravel_inputs = True
>>> out = merge.run()
>>> out.outputs.out
[1, 2, 5, 3]
```

```
>>> merge = Merge(1)
>>> merge.inputs.in1 = [1, [2, 5], 3]
>>> merge.inputs.no_flatten = True
>>> out = merge.run()
>>> out.outputs.out
[[1, [2, 5], 3]]
```


Inputs:

```
[Mandatory]

[Optional]
axis: ('vstack' or 'hstack', nipyype default value: vstack)
      direction in which to merge, hstack requires same number of elements
      in each input
ignore_exception: (a boolean, nipyype default value: False)
      Print an error message instead of throwing an exception in case the
      interface fails to run
no_flatten: (a boolean, nipyype default value: False)
      append to outlist instead of extending in vstack mode
ravel_inputs: (a boolean, nipyype default value: False)
      ravel inputs when no_flatten is False
```

Outputs:

```
out: (a list of items which are any value)
      Merged output
```

Rename

[Link to code](#)

Change the name of a file based on a mapped format string.

To use additional inputs that will be defined at run-time, the class constructor must be called with the format template, and the fields identified will become inputs to the interface.

Additionally, you may set the parse_string input, which will be run over the input filename with a regular expressions search, and will fill in additional input fields from matched groups. Fields set with inputs have precedence over fields filled in with the regexp match.

Examples

```
>>> from nipyype.interfaces.utility import Rename
>>> rename1 = Rename()
>>> rename1.inputs.in_file = "zstat1.nii.gz"
>>> rename1.inputs.format_string = "Faces-Scenes.nii.gz"
>>> res = rename1.run()
>>> res.outputs.out_file
'Faces-Scenes.nii.gz'
```

```
>>> rename2 = Rename(format_string="% (subject_id) s_func_run% (run) 02d")
>>> rename2.inputs.in_file = "functional.nii"
>>> rename2.inputs.keep_ext = True
>>> rename2.inputs.subject_id = "subj_201"
>>> rename2.inputs.run = 2
>>> res = rename2.run()
>>> res.outputs.out_file
'subj_201_func_run02.nii'
```

```
>>> rename3 = Rename(format_string="% (subject_id) s_% (seq) s_run% (run) 02d.nii")
>>> rename3.inputs.in_file = "func_epi_1_1.nii"
>>> rename3.inputs.parse_string = "func_(?P<seq>\\w*)_.*"
>>> rename3.inputs.subject_id = "subj_201"
>>> rename3.inputs.run = 2
>>> res = rename3.run()
```

```
>>> res.outputs.out_file
'subj_201_epi_run02.nii'
```

Inputs:

```
[Mandatory]
format_string: (a unicode string)
    Python formatting string for output template
in_file: (an existing file name)
    file to rename

[Optional]
keep_ext: (a boolean)
    Keep in_file extension, replace non-extension component of name
parse_string: (a unicode string)
    Python regexp parse string to define replacement inputs
use_fullpath: (a boolean, nipy default value: False)
    Use full path as input to regex parser
```

Outputs:

```
out_file: (a file name)
    softlink to original file with new name
```

Select

[Link to code](#)

Basic interface class to select specific elements from a list

Examples

```
>>> from nipy.interfaces.utility import Select
>>> sl = Select()
>>> _ = sl.inputs.set(inlist=[1, 2, 3, 4, 5], index=[3])
>>> out = sl.run()
>>> out.outputs.out
~
```

```
>>> _ = sl.inputs.set(inlist=[1, 2, 3, 4, 5], index=[3, 4])
>>> out = sl.run()
>>> out.outputs.out
[4, 5]
```

Inputs:

```
[Mandatory]
index: (a list of items which are an integer (int or long))
    0-based indices of values to choose
inlist: (a list of items which are any value)
    list of values to choose from

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

Outputs:

```
out: (a list of items which are any value)
      list of selected values
```

Split

[Link to code](#)

Basic interface class to split lists into multiple outputs

Examples

```
>>> from nipyre.interfaces.utility import Split
>>> sp = Split()
>>> _ = sp.inputs.set(inlist=[1, 2, 3], splits=[2, 1])
>>> out = sp.run()
>>> out.outputs.out1
[1, 2]
```

Inputs:

```
[Mandatory]
inlist: (a list of items which are any value)
        list of values to split
splits: (a list of items which are an integer (int or long))
        Number of outputs in each split - should add to number of inputs

[Optional]
ignore_exception: (a boolean, nipyre default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
squeeze: (a boolean, nipyre default value: False)
          unfold one-element splits removing the list
```

Outputs:

```
None
```

interfaces.utility.csv

CSVReader

[Link to code](#)

Examples

```
>>> reader = CSVReader()
>>> reader.inputs.in_file = 'noHeader.csv'
>>> out = reader.run()
>>> out.outputs.column_0 == ['foo', 'bar', 'baz']
True
>>> out.outputs.column_1 == ['hello', 'world', 'goodbye']
True
>>> out.outputs.column_2 == ['300.1', '5', '0.3']
True
```

```
>>> reader = CSVReader()
>>> reader.inputs.in_file = 'header.csv'
>>> reader.inputs.header = True
>>> out = reader.run()
>>> out.outputs.files == ['foo', 'bar', 'baz']
True
>>> out.outputs.labels == ['hello', 'world', 'goodbye']
True
>>> out.outputs.erosion == ['300.1', '5', '0.3']
True
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         Input comma-seperated value (CSV) file

[Optional]
header: (a boolean, nipyne default value: False)
        True if the first line is a column header
```

Outputs:

None

interfaces.utility.wrappers

Function

[Link to code](#)

Runs arbitrary function as an interface

Examples

```
>>> func = 'def func(arg1, arg2=5): return arg1 + arg2'
>>> fi = Function(input_names=['arg1', 'arg2'], output_names=['out'])
>>> fi.inputs.function_str = func
>>> res = fi.run(arg1=1)
>>> res.outputs.out
~
```

Inputs:

```
[Mandatory]
function_str: (a unicode string)
              code for function

[Optional]
ignore_exception: (a boolean, nipyne default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
```

Outputs:

None

interfaces.vista.vista

Vnifti2Image

[Link to code](#)

Wraps command **vnifti2image**

Convert a nifti file into a vista file.

Example

```
>>> vimage = Vnifti2Image()
>>> vimage.inputs.in_file = 'image.nii'
>>> vimage.cmdline
'vnifti2image -in image.nii -out image.v'
>>> vimage.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         in file
         flag: -in %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
attributes: (an existing file name)
            attribute file
            flag: -attr %s, position: 2
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a value
         of class 'str', nipype default value: {})
         Environment variables
ignore_exception: (a boolean, nipype default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
```

```
out_file: (a file name)
    output data file
    flag: -out %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    Output vista file
```

VtoMat

[Link to code](#)Wraps command **vtomat**

Convert a nifti file into a vista file.

Example

```
>>> vimage = VtoMat()
>>> vimage.inputs.in_file = 'image.v'
>>> vimage.cmdline
'vtomat -in image.v -out image.mat'
>>> vimage.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    in file
    flag: -in %s, position: 1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
out_file: (a file name)
    output mat file
    flag: -out %s, position: -1
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
out_file: (an existing file name)
    Output mat file
```

Bru2

[Link to code](#)

Wraps command **Bru2**

Uses bru2nii's Bru2 to convert Bruker files

Examples

```
>>> from nipy.interfaces.bru2nii import Bru2
>>> converter = Bru2()
>>> converter.inputs.input_dir = "brukerdir"
>>> converter.cmdline
'Bru2 -o ../nipy/testing/data/brukerdir brukerdir'
```

Inputs:

```
[Mandatory]
input_dir: (an existing directory name)
    Input Directory
    flag: %s, position: -1

[Optional]
actual_size: (a boolean)
    Keep actual size - otherwise x10 scale so animals match human.
    flag: -a
append_protocol_name: (a boolean)
    Append protocol name to output filename.
    flag: -p
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
force_conversion: (a boolean)
```

```
    Force conversion of localizers images (multiple slice orientations).
    flag: -f
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
output_filename: (a unicode string)
    Output filename ('.nii' will be appended)
    flag: -o %s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
nii_file: (an existing file name)
```


C3dAffineTool

[Link to code](#)

Wraps command **c3d_affine_tool**

Converts fsl-style Affine registration into ANTS compatible itk format

Example

```
>>> from nipy.interfaces.c3 import C3dAffineTool
>>> c3 = C3dAffineTool()
>>> c3.inputs.source_file = 'cmatrix.mat'
>>> c3.inputs.itk_transform = 'affine.txt'
>>> c3.inputs.fsl2ras = True
>>> c3.cmdline
'c3d_affine_tool -src cmatrix.mat -fsl2ras -oitk affine.txt'
```

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
fsl2ras: (a boolean)
    flag: -fsl2ras, position: 4
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
itk_transform: (a boolean or a file name)
    Export ITK transform.
    flag: -oitk %s, position: 5
```

```
reference_file: (an existing file name)
    flag: -ref %s, position: 1
source_file: (an existing file name)
    flag: -src %s, position: 2
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
transform_file: (an existing file name)
    flag: %s, position: 3
```

Outputs:

```
itk_transform: (an existing file name)
```

Dcm2nii

[Link to code](#)

Wraps command **dcm2nii**

Uses MRICron's dcm2nii to convert dicom files

Examples

```
>>> from nipy.interfaces.dcm2nii import Dcm2nii
>>> converter = Dcm2nii()
>>> converter.inputs.source_names = ['functional_1.dcm', 'functional_2.dcm']
>>> converter.inputs.gzip_output = True
>>> converter.inputs.output_dir = '.'
>>> converter.cmdline
'dcm2nii -a y -c y -b config.ini -v y -d y -e y -g y -i n -n y -o . -p y -x n -f_
↪n functional_1.dcm'
```

Inputs:

```
[Mandatory]
source_dir: (an existing directory name)
    flag: %s, position: -l
    mutually_exclusive: source_names
source_names: (a list of items which are an existing file name)
    flag: %s, position: -l
    mutually_exclusive: source_dir

[Optional]
anonymize: (a boolean, nipy default value: True)
    Remove identifying information
    flag: -a
args: (a unicode string)
    Additional parameters to the command
    flag: %s
collapse_folders: (a boolean, nipy default value: True)
    Collapse input folders
```

```
    flag: -c
config_file: (an existing file name)
    Load settings from specified inifile
    flag: -b %s
convert_all_pars: (a boolean, nipy default value: True)
    Convert every image in directory
    flag: -v
date_in_filename: (a boolean, nipy default value: True)
    Date in filename
    flag: -d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
events_in_filename: (a boolean, nipy default value: True)
    Events (series/acq) in filename
    flag: -e
gzip_output: (a boolean, nipy default value: False)
    Gzip output (.gz)
    flag: -g
id_in_filename: (a boolean, nipy default value: False)
    ID in filename
    flag: -i
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
nii_output: (a boolean, nipy default value: True)
    Save as .nii - if no, create .hdr/.img pair
    flag: -n
output_dir: (an existing directory name)
    Output dir - if unspecified, source directory is used
    flag: -o %s
protocol_in_filename: (a boolean, nipy default value: True)
    Protocol in filename
    flag: -p
reorient: (a boolean)
    Reorient image to nearest orthogonal
    flag: -r
reorient_and_crop: (a boolean, nipy default value: False)
    Reorient and crop 3D images
    flag: -x
source_in_filename: (a boolean, nipy default value: False)
    Source filename
    flag: -f
spm_analyze: (a boolean)
    SPM2/Analyze not SPM5/NIfTI
    flag: -s
    mutually_exclusive: nii_output
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

```
bvals: (a list of items which are an existing file name)
bvecs: (a list of items which are an existing file name)
converted_files: (a list of items which are an existing file name)
```

```
reoriented_and_cropped_files: (a list of items which are an existing
    file name)
reoriented_files: (a list of items which are an existing file name)
```

Dcm2niix

[Link to code](#)

Wraps command **dcm2niix**

Uses Chris Rorden's dcm2niix to convert dicom files

Examples

```
>>> from nipy.interfaces.dcm2nii import Dcm2niix
>>> converter = Dcm2niix()
>>> converter.inputs.source_names = ['functional_1.dcm', 'functional_2.dcm']
>>> converter.inputs.compress = 'i'
>>> converter.inputs.single_file = True
>>> converter.inputs.output_dir = '.'
>>> converter.cmdline
'dcm2niix -b y -z i -x n -t n -m n -f %t%p -o . -s y -v n functional_1.dcm'
```

```
>>> flags = '-'.join([val.strip() + ' ' for val in sorted('-'.join(converter.
↳cmdline.split()[1:-1]).split('-'))])
>>> flags
'-b y -f %t%p -m n -o . -s y -t n -v n -x n -z i '
```

Inputs:

```
[Mandatory]
source_dir: (an existing directory name)
    flag: %s, position: -1
    mutually_exclusive: source_names
source_names: (a list of items which are an existing file name)
    flag: %s, position: -1
    mutually_exclusive: source_dir

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
bids_format: (a boolean, nipy default value: True)
    Create a BIDS sidecar file
    flag: -b
compress: ('y' or 'i' or 'n', nipy default value: i)
    Gzip compress images - [y=pigz, i=internal, n=no]
    flag: -z %s
crop: (a boolean, nipy default value: False)
    Crop 3D T1 acquisitions
    flag: -x
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
has_private: (a boolean, nipy default value: False)
    Flag if text notes includes private patient details
```

```
    flag: -t
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
merge_imgs: (a boolean, nipy default value: False)
    merge 2D slices from same series
    flag: -m
out_filename: (a unicode string, nipy default value: %t%p)
    Output filename
    flag: -f %s
output_dir: (an existing directory name)
    Output directory
    flag: -o %s
single_file: (a boolean, nipy default value: False)
    Convert only one image (filename as last input
    flag: -s
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
verbose: (a boolean, nipy default value: False)
    Verbose output
    flag: -v
```

Outputs:

```
bids: (a list of items which are an existing file name)
bvals: (a list of items which are an existing file name)
bvecs: (a list of items which are an existing file name)
converted_files: (a list of items which are an existing file name)
```

CopyMeta

[Link to code](#)

Copy meta data from one Nifti file to another. Useful for preserving meta data after some processing steps.

Inputs:

```
[Mandatory]
dest_file: (an existing file name)
src_file: (an existing file name)

[Optional]
exclude_classes: (a list of items which are any value)
                  List of meta data classifications to exclude
include_classes: (a list of items which are any value)
                  List of specific meta data classifications to include. If not
                  specified include everything.
```

Outputs:

```
dest_file: (an existing file name)
```

DcmStack

[Link to code](#)

Create one Nifti file from a set of DICOM files. Can optionally embed meta data.

Example

```
>>> from nipy.interfaces.dcmstack import DcmStack
>>> stacker = DcmStack()
>>> stacker.inputs.dicom_files = 'path/to/series/'
>>> stacker.run()
>>> result.outputs.out_file
'/path/to/cwd/sequence.nii.gz'
```

Inputs:

```
[Mandatory]
dicom_files: (a list of items which are an existing file name or an
             existing directory name or a unicode string)

[Optional]
embed_meta: (a boolean)
             Embed DICOM meta data into result
exclude_regexes: (a list of items which are any value)
                 Meta data to exclude, supplementing any default exclude filters
force_read: (a boolean, nipy default value: True)
             Force reading files without DICM marker
include_regexes: (a list of items which are any value)
                 Meta data to include, overriding any exclude filters
out_ext: (a unicode string, nipy default value: .nii.gz)
          Determines output file type
out_format: (a unicode string)
             String which can be formatted with meta data to create the output
             filename(s)
out_path: (a directory name)
           output path, current working directory if not set
```

Outputs:

```
out_file: (an existing file name)
```

GroupAndStack

[Link to code](#)

Create (potentially) multiple Nifti files for a set of DICOM files.

Inputs:

```
[Mandatory]
dicom_files: (a list of items which are an existing file name or an
             existing directory name or a unicode string)

[Optional]
embed_meta: (a boolean)
             Embed DICOM meta data into result
exclude_regexes: (a list of items which are any value)
                 Meta data to exclude, supplementing any default exclude filters
force_read: (a boolean, nipy default value: True)
             Force reading files without DICM marker
include_regexes: (a list of items which are any value)
                 Meta data to include, overriding any exclude filters
out_ext: (a unicode string, nipy default value: .nii.gz)
          Determines output file type
out_format: (a unicode string)
             String which can be formatted with meta data to create the output
             filename(s)
out_path: (a directory name)
           output path, current working directory if not set
```

Outputs:

```
out_list: (a list of items which are any value)
           List of output nifti files
```


LookupMeta

[Link to code](#)

Lookup meta data values from a Nifti with embedded meta data.

Example

```
>>> from nipy.interfaces import dcmstack
>>> lookup = dcmstack.LookupMeta()
>>> lookup.inputs.in_file = 'functional.nii'
>>> lookup.inputs.meta_keys = {'RepetitionTime' : 'TR',
    ↪      'EchoTime' : 'TE'}
>>> result = lookup.run()
>>> result.outputs.TR
9500.0
>>> result.outputs.TE
95.0
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    The input Nifti file
meta_keys: (a list of items which are any value or a dictionary with
    keys which are any value and with values which are any value)
    List of meta data keys to lookup, or a dict where keys specify the
    meta data keys to lookup and the values specify the output names

[Optional]
```

Outputs:

None

MergeNifti

[Link to code](#)

Merge multiple Nifti files into one. Merges together meta data extensions as well.

Inputs:

```
[Mandatory]
in_files: (a list of items which are any value)
    List of Nifti files to merge

[Optional]
merge_dim: (an integer (int or long))
    Dimension to merge along. If not specified, the last singular or
    non-existent dimension is used.
out_ext: (a unicode string, nipy default value: .nii.gz)
    Determines output file type
out_format: (a unicode string)
    String which can be formatted with meta data to create the output
    filename(s)
out_path: (a directory name)
    output path, current working directory if not set
sort_order: (a unicode string or a list of items which are any value)
    One or more meta data keys to sort files by.
```

Outputs:

```
out_file: (an existing file name)
    Merged Nifti file
```

NiftiGeneratorBase

[Link to code](#)

Base class for interfaces that produce Nifti files, potentially with embedded meta data.

Inputs:

```
[Mandatory]

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

Outputs:

```
None
```

SplitNifti

[Link to code](#)

Split one Nifti file into many along the specified dimension. Each result has an updated meta data extension as well.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    Nifti file to split

[Optional]
out_ext: (a unicode string, nipy default value: .nii.gz)
    Determines output file type
out_format: (a unicode string)
    String which can be formatted with meta data to create the output
    filename(s)
out_path: (a directory name)
    output path, current working directory if not set
split_dim: (an integer (int or long))
    Dimension to split along. If not specified, the last dimension is
    used.
```

Outputs:

```
out_list: (a list of items which are an existing file name)
    Split Nifti files
```

make_key_func()

[Link to code](#)

`sanitize_path_comp()`

[Link to code](#)

SlicerCommandLine

[Link to code](#)

Wraps command **Slicer3**

Experimental Slicer wrapper. Work in progress.

Inputs:

```
[Mandatory]

[Optional]
args: (a unicode string)
    Additional parameters to the command
    flag: %s
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
module: (a unicode string)
    name of the Slicer command line module you want to use
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
```

Outputs:

None

DataFinder

[Link to code](#)

Search for paths that match a given regular expression. Allows a less proscriptive approach to gathering input files compared to DataGrabber. Will recursively search any subdirectories by default. This can be limited with the min/max depth options. Matched paths are available in the output ‘out_paths’. Any named groups of captured text from the regular expression are also available as outputs of the same name.

Examples

```
>>> from nipype.interfaces.io import DataFinder
>>> df = DataFinder()
>>> df.inputs.root_paths = '.'
>>> df.inputs.match_regex = '.*/(?P<series_dir>.*(qT1|ep2d_fid_T1).+)/(?P
↪<basename>.+)\.nii.gz'
>>> result = df.run()
>>> result.outputs.out_paths
['./027-ep2d_fid_T1_Gd4/acquisition.nii.gz',
 './018-ep2d_fid_T1_Gd2/acquisition.nii.gz',
 './016-ep2d_fid_T1_Gd1/acquisition.nii.gz',
 './013-ep2d_fid_T1_pre/acquisition.nii.gz']
>>> result.outputs.series_dir
['027-ep2d_fid_T1_Gd4',
 '018-ep2d_fid_T1_Gd2',
 '016-ep2d_fid_T1_Gd1',
 '013-ep2d_fid_T1_pre']
>>> result.outputs.basename
['acquisition',
 'acquisition',
 'acquisition',
 'acquisition']
```

Inputs:

```
[Mandatory]
root_paths: (a list of items which are any value or a unicode string)
```

```
[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
ignore_regexes: (a list of items which are any value)
    List of regular expressions, if any match the path it will be
    ignored.
match_regex: (a unicode string, nipy default value: (.+))
    Regular expression for matching paths.
max_depth: (an integer (int or long))
    The maximum depth to search beneath the root_paths
min_depth: (an integer (int or long))
    The minimum depth to search beneath the root paths
unpack_single: (a boolean, nipy default value: False)
    Unpack single results from list
```

Outputs:

None

DataGrabber

[Link to code](#)

Generic datagrabber module that wraps around glob in an intelligent way for neuroimaging tasks to grab files

Attention: Doesn't support directories currently

Examples

```
>>> from nipy.interfaces.io import DataGrabber
```

Pick all files from current directory

```
>>> dg = DataGrabber()
>>> dg.inputs.template = '*'
```

Pick file foo/foo.nii from current directory

```
>>> dg.inputs.template = '%s/%s.dcm'
>>> dg.inputs.template_args['outfiles']=[['dicomdir', '123456-1-1.dcm']]
```

Same thing but with dynamically created fields

```
>>> dg = DataGrabber(infields=['arg1', 'arg2'])
>>> dg.inputs.template = '%s/%s.nii'
>>> dg.inputs.arg1 = 'foo'
>>> dg.inputs.arg2 = 'foo'
```

however this latter form can be used with iterables and iterfield in a pipeline.

Dynamically created, user-defined input and output fields

```
>>> dg = DataGrabber(infields=['sid'], outfields=['func', 'struct', 'ref'])
>>> dg.inputs.base_directory = '.'
>>> dg.inputs.template = '%s/%s.nii'
>>> dg.inputs.template_args['func'] = [['sid', ['f3', 'f5']]]
>>> dg.inputs.template_args['struct'] = [['sid', ['struct']]]
```



```
>>> dg.inputs.template_args['ref'] = [['sid', 'ref']]
>>> dg.inputs.sid = 's1'
```

Change the template only for output field struct. The rest use the general template

```
>>> dg.inputs.field_template = dict(struct='%s/struct.nii')
>>> dg.inputs.template_args['struct'] = [['sid']]
```

Inputs:

```
[Mandatory]
sort_filelist: (a boolean)
    Sort the filelist that matches the template
template: (a unicode string)
    Layout used to get files. relative to base directory if defined

[Optional]
base_directory: (an existing directory name)
    Path to the base directory consisting of subject data.
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
raise_on_empty: (a boolean, nipype default value: True)
    Generate exception if list is empty for a given field
template_args: (a dictionary with keys which are a unicode string and
    with values which are a list of items which are a list of items
    which are any value)
    Information to plug into template
```

Outputs:

None

DataSink

[Link to code](#)

Generic datasink module to store structured outputs

Primarily for use within a workflow. This interface allows arbitrary creation of input attributes. The names of these attributes define the directory structure to create for storage of the files or directories.

The attributes take the following form:

string[.[@]]string[.[@]]string] ...

where parts between [] are optional.

An attribute such as contrasts.@con will create a 'contrasts' directory to store the results linked to the attribute.

If the @ is left out, such as in 'contrasts.con', a subdirectory 'con' will be created under 'contrasts'.

the general form of the output is:

```
'base_directory/container/parameterization/destloc/filename'
```

```
destloc = string[.[@]]string[.[@]]string] and
filename comesfrom the input to the connect statement.
```

Warning: This is not a thread-safe node because it can write to a common shared location. It will not complain when it overwrites a file.

Note: If both substitutions and regexp_substitutions are used, then substitutions are applied first followed by

regex_substitutions.

This interface **cannot** be used in a MapNode as the inputs are defined only when the connect statement is executed.

Examples

```
>>> ds = DataSink()
>>> ds.inputs.base_directory = 'results_dir'
>>> ds.inputs.container = 'subject'
>>> ds.inputs.structural = 'structural.nii'
>>> setattr(ds.inputs, 'contrasts.@con', ['cont1.nii', 'cont2.nii'])
>>> setattr(ds.inputs, 'contrasts.alt', ['cont1a.nii', 'cont2a.nii'])
>>> ds.run()
```

To use DataSink in a MapNode, its inputs have to be defined at the time the interface is created.

```
>>> ds = DataSink(infields=['contrasts.@con'])
>>> ds.inputs.base_directory = 'results_dir'
>>> ds.inputs.container = 'subject'
>>> ds.inputs.structural = 'structural.nii'
>>> setattr(ds.inputs, 'contrasts.@con', ['cont1.nii', 'cont2.nii'])
>>> setattr(ds.inputs, 'contrasts.alt', ['cont1a.nii', 'cont2a.nii'])
>>> ds.run()
```

Inputs:

```
[Mandatory]

[Optional]
_outputs: (a dictionary with keys which are a unicode string and with
          values which are any value, nipyre default value: {})
base_directory: (a directory name)
                Path to the base directory for storing data.
bucket: (any value)
        Boto3 S3 bucket for manual override of bucket
container: (a unicode string)
           Folder within base directory in which to store output
creds_path: (a unicode string)
            Filepath to AWS credentials file for S3 bucket access; if not
            specified, the credentials will be taken from the AWS_ACCESS_KEY_ID
            and AWS_SECRET_ACCESS_KEY environment variables
encrypt_bucket_keys: (a boolean)
                    Flag indicating whether to use S3 server-side AES-256 encryption
ignore_exception: (a boolean, nipyre default value: False)
                 Print an error message instead of throwing an exception in case the
                 interface fails to run
local_copy: (a unicode string)
            Copy files locally as well as to S3 bucket
parameterization: (a boolean, nipyre default value: True)
                  store output in parametrized structure
regex_substitutions: (a list of items which are a tuple of the form:
                     (a unicode string, a unicode string))
                     List of 2-tuples reflecting a pair of a Python regexp pattern and a
                     replacement string. Invoked after string `substitutions`
remove_dest_dir: (a boolean, nipyre default value: False)
                 remove dest directory when copying dirs
strip_dir: (a directory name)
```

```

    path to strip out of filename
substitutions: (a list of items which are a tuple of the form: (a
                  unicode string, a unicode string))
                List of 2-tuples reflecting string to substitute and string to
                replace it with

```

Outputs:

```

out_file: (any value)
          datasink output

```

FreeSurferSource

[Link to code](#)

Generates freesurfer subject info from their directories

Examples

```

>>> from nipy.interfaces.io import FreeSurferSource
>>> fs = FreeSurferSource()
>>> #fs.inputs.subjects_dir = '.'
>>> fs.inputs.subject_id = 'PWS04'
>>> res = fs.run()

```

```

>>> fs.inputs.hemi = 'lh'
>>> res = fs.run()

```

Inputs:

```

[Mandatory]
subject_id: (a unicode string)
             Subject name for whom to retrieve data
subjects_dir: (a directory name)
               Freesurfer subjects directory.

[Optional]
hemi: ('both' or 'lh' or 'rh', nipy default value: both)
       Selects hemisphere specific outputs
ignore_exception: (a boolean, nipy default value: False)
                   Print an error message instead of throwing an exception in case the
                   interface fails to run

```

Outputs:

```

BA_stats: (a list of items which are an existing file name)
           Brodmann Area statistics files
T1: (an existing file name)
     Intensity normalized whole-head volume
annot: (a list of items which are an existing file name)
        Surface annotation files
aparc_a2009s_stats: (a list of items which are an existing file name)
                    Aparc a2009s parcellation statistics files
aparc_aseg: (a list of items which are an existing file name)
             Aparc parcellation projected into aseg volume
aparc_stats: (a list of items which are an existing file name)
              Aparc parcellation statistics files

```

```
area_pial: (a list of items which are an existing file name)
    Mean area of triangles each vertex on the pial surface is associated
    with
aseg: (an existing file name)
    Volumetric map of regions from automatic segmentation
aseg_stats: (a list of items which are an existing file name)
    Automated segmentation statistics file
avg_curv: (a list of items which are an existing file name)
    Average atlas curvature, sampled to subject
brain: (an existing file name)
    Intensity normalized brain-only volume
brainmask: (an existing file name)
    Skull-stripped (brain-only) volume
curv: (a list of items which are an existing file name)
    Maps of surface curvature
curv_pial: (a list of items which are an existing file name)
    Curvature of pial surface
curv_stats: (a list of items which are an existing file name)
    Curvature statistics files
entorhinal_exvivo_stats: (a list of items which are an existing file
    name)
    Entorhinal exvivo statistics files
filled: (an existing file name)
    Subcortical mass volume
graymid: (a list of items which are an existing file name)
    Graymid/midthickness surface meshes
inflated: (a list of items which are an existing file name)
    Inflated surface meshes
jacobian_white: (a list of items which are an existing file name)
    Distortion required to register to spherical atlas
label: (a list of items which are an existing file name)
    Volume and surface label files
norm: (an existing file name)
    Normalized skull-stripped volume
nu: (an existing file name)
    Non-uniformity corrected whole-head volume
orig: (an existing file name)
    Base image conformed to Freesurfer space
pial: (a list of items which are an existing file name)
    Gray matter/pia mater surface meshes
rawavg: (an existing file name)
    Volume formed by averaging input images
ribbon: (a list of items which are an existing file name)
    Volumetric maps of cortical ribbons
smoothwm: (a list of items which are an existing file name)
    Smoothed original surface meshes
sphere: (a list of items which are an existing file name)
    Spherical surface meshes
sphere_reg: (a list of items which are an existing file name)
    Spherical registration file
sulc: (a list of items which are an existing file name)
    Surface maps of sulcal depth
thickness: (a list of items which are an existing file name)
    Surface maps of cortical thickness
volume: (a list of items which are an existing file name)
    Surface maps of cortical volume
white: (a list of items which are an existing file name)
    White/gray matter surface meshes
```

```

wm: (an existing file name)
    Segmented white-matter volume
wmparc: (an existing file name)
    Aparc parcellation projected into subcortical white matter
wmparc_stats: (a list of items which are an existing file name)
    White matter parcellation statistics file

```

IOBase

[Link to code](#)

Inputs:

```

[Mandatory]

[Optional]
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run

```

Outputs:

```

None

```

JSONFileGrabber

[Link to code](#)

Datagrabber interface that loads a json file and generates an output for every first-level object

Example

```

>>> import pprint
>>> from nipy.interfaces.io import JSONFileGrabber
>>> jsonSource = JSONFileGrabber()
>>> jsonSource.inputs.defaults = {'param1': 'overrideMe', 'param3': 1.0}
>>> res = jsonSource.run()
>>> pprint.pprint(res.outputs.get())
{'param1': 'overrideMe', 'param3': 1.0}
>>> jsonSource.inputs.in_file = 'jsongrabber.txt'
>>> res = jsonSource.run()
>>> pprint.pprint(res.outputs.get())
{'param1': 'exampleStr', 'param2': 4, 'param3': 1.0}

```

Inputs:

```

[Mandatory]

[Optional]
defaults: (a dictionary with keys which are any value and with values
    which are any value)
    JSON dictionary that sets default outputvalues, overridden by values
    found in in_file
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run

```

```
in_file: (an existing file name)
        JSON source file
```

Outputs:

```
None
```

JSONFileSink

[Link to code](#)

Very simple frontend for storing values into a JSON file. Entries already existing in `in_dict` will be overridden by matching entries dynamically added as inputs.

Warning: This is not a thread-safe node because it can write to a common shared location. It will not complain when it overwrites a file.

```
>>> jsonsink = JSONFileSink(input_names=['subject_id',
...                                   'some_measurement'])
>>> jsonsink.inputs.subject_id = 's1'
>>> jsonsink.inputs.some_measurement = 11.4
>>> jsonsink.run()
```

Using a dictionary as input:

```
>>> dictsink = JSONFileSink()
>>> dictsink.inputs.in_dict = {'subject_id': 's1',
...                            'some_measurement': 11.4}
>>> dictsink.run()
```

Inputs:

```
[Mandatory]

[Optional]
_outputs: (a dictionary with keys which are any value and with values
          which are any value, nipyne default value: {})
ignore_exception: (a boolean, nipyne default value: False)
                  Print an error message instead of throwing an exception in case the
                  interface fails to run
in_dict: (a dictionary with keys which are any value and with values
          which are any value, nipyne default value: {})
          input JSON dictionary
out_file: (a file name)
          JSON sink file
```

Outputs:

```
out_file: (a file name)
          JSON sink file
```

MySQLSink

[Link to code](#)

Very simple frontend for storing values into MySQL database.

Examples

```
>>> sql = MySQLSink(input_names=['subject_id', 'some_measurement'])
>>> sql.inputs.database_name = 'my_database'
>>> sql.inputs.table_name = 'experiment_results'
>>> sql.inputs.username = 'root'
>>> sql.inputs.password = 'secret'
>>> sql.inputs.subject_id = 's1'
>>> sql.inputs.some_measurement = 11.4
>>> sql.run()
```

Inputs:

```
[Mandatory]
config: (a file name)
    MySQL Options File (same format as my.cnf)
    mutually_exclusive: host
database_name: (a unicode string)
    Otherwise known as the schema name
host: (a unicode string, nipype default value: localhost)
    mutually_exclusive: config
    requires: username, password
table_name: (a unicode string)

[Optional]
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
password: (a unicode string)
username: (a unicode string)
```

Outputs:

```
None
```

S3DataGrabber

[Link to code](#)

Generic datagrabber module that wraps around glob in an intelligent way for neuroimaging tasks to grab files from Amazon S3

Works exactly like DataGrabber, except, you must specify an S3 “bucket” and “bucket_path” to search for your data and a “local_directory” to store the data. “local_directory” should be a location on HDFS for Spark jobs. Additionally, “template” uses regex style formatting, rather than the glob-style found in the original DataGrabber.

Inputs:

```
[Mandatory]
bucket: (a unicode string)
    Amazon S3 bucket where your data is stored
sort_filelist: (a boolean)
    Sort the filelist that matches the template
template: (a unicode string)
    Layout used to get files. Relative to bucket_path if defined. Uses
    regex rather than glob style formatting.

[Optional]
anon: (a boolean, nipype default value: False)
```

```
    Use anonymous connection to s3. If this is set to True, boto may
    print a urlopen error, but this does not prevent data from being
    downloaded.
bucket_path: (a unicode string, nipyte default value: )
    Location within your bucket for subject data.
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
local_directory: (an existing directory name)
    Path to the local directory for subject data to be downloaded and
    accessed. Should be on HDFS for Spark jobs.
raise_on_empty: (a boolean, nipyte default value: True)
    Generate exception if list is empty for a given field
region: (a unicode string, nipyte default value: us-east-1)
    Region of s3 bucket
template_args: (a dictionary with keys which are a unicode string and
    with values which are a list of items which are a list of items
    which are any value)
    Information to plug into template
```

Outputs:

None

SQLiteSink

[Link to code](#)

Very simple frontend for storing values into SQLite database.

Warning: This is not a thread-safe node because it can write to a common shared location. It will not complain when it overwrites a file.

Examples

```
>>> sql = SQLiteSink(input_names=['subject_id', 'some_measurement'])
>>> sql.inputs.database_file = 'my_database.db'
>>> sql.inputs.table_name = 'experiment_results'
>>> sql.inputs.subject_id = 's1'
>>> sql.inputs.some_measurement = 11.4
>>> sql.run()
```

Inputs:

```
[Mandatory]
database_file: (an existing file name)
table_name: (a unicode string)

[Optional]
ignore_exception: (a boolean, nipyte default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
```

Outputs:

None

SSHDataGrabber

[Link to code](#)

Extension of DataGrabber module that downloads the file list and optionally the files from a SSH server. The SSH operation must not need user and password so an SSH agent must be active in where this module is being run.

Attention: Doesn't support directories currently

Examples

```
>>> from nipy.interfaces.io import SSHDataGrabber
>>> dg = SSHDataGrabber()
>>> dg.inputs.hostname = 'test.rebex.net'
>>> dg.inputs.user = 'demo'
>>> dg.inputs.password = 'password'
>>> dg.inputs.base_directory = 'pub/example'
```

Pick all files from the base directory

```
>>> dg.inputs.template = '*'
```

Pick all files starting with “s” and a number from current directory

```
>>> dg.inputs.template_expression = 'regexp'
>>> dg.inputs.template = 'pop[0-9].*'
```

Same thing but with dynamically created fields

```
>>> dg = SSHDataGrabber(infields=['arg1', 'arg2'])
>>> dg.inputs.hostname = 'test.rebex.net'
>>> dg.inputs.user = 'demo'
>>> dg.inputs.password = 'password'
>>> dg.inputs.base_directory = 'pub'
>>> dg.inputs.template = '%s/%s.txt'
>>> dg.inputs.arg1 = 'example'
>>> dg.inputs.arg2 = 'foo'
```

however this latter form can be used with iterables and iterfield in a pipeline.

Dynamically created, user-defined input and output fields

```
>>> dg = SSHDataGrabber(infields=['sid'], outfields=['func', 'struct', 'ref'])
>>> dg.inputs.hostname = 'myhost.com'
>>> dg.inputs.base_directory = '/main_folder/my_remote_dir'
>>> dg.inputs.template_args['func'] = [['sid', ['f3', 'f5']]]
>>> dg.inputs.template_args['struct'] = [['sid', ['struct']]]
>>> dg.inputs.template_args['ref'] = [['sid', 'ref']]
>>> dg.inputs.sid = 's1'
```

Change the template only for output field struct. The rest use the general template

```
>>> dg.inputs.field_template = dict(struct='%s/struct.nii')
>>> dg.inputs.template_args['struct'] = [['sid']]
```

Inputs:

```
[Mandatory]
base_directory: (a unicode string)
    Path to the base directory consisting of subject data.
hostname: (a unicode string)
    Server hostname.
sort_filelist: (a boolean)
    Sort the filelist that matches the template
template: (a unicode string)
    Layout used to get files. relative to base directory if defined

[Optional]
download_files: (a boolean, nipyre default value: True)
    If false it will return the file names without downloading them
ignore_exception: (a boolean, nipyre default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
password: (a string)
    Server password.
raise_on_empty: (a boolean, nipyre default value: True)
    Generate exception if list is empty for a given field
ssh_log_to_file: (a unicode string, nipyre default value: )
    If set SSH commands will be logged to the given file
template_args: (a dictionary with keys which are a unicode string and
    with values which are a list of items which are a list of items
    which are any value)
    Information to plug into template
template_expression: ('fnmatch' or 'regex', nipyre default value:
    fnmatch)
    Use either fnmatch or regex to express templates
username: (a unicode string)
    Server username.
```

Outputs:

None

SelectFiles

[Link to code](#)

Flexibly collect data from disk to feed into workflows.

This interface uses the {}-based string formatting syntax to plug values (possibly known only at workflow execution time) into string templates and collect files from persistent storage. These templates can also be combined with glob wildcards. The field names in the formatting template (i.e. the terms in braces) will become inputs fields on the interface, and the keys in the templates dictionary will form the output fields.

Examples

```
>>> import pprint
>>> from nipyre import SelectFiles, Node
>>> templates={"T1": "{subject_id}/struct/T1.nii",
...           "epi": "{subject_id}/func/f[0, 1].nii"}
>>> dg = Node(SelectFiles(templates), "selectfiles")
>>> dg.inputs.subject_id = "subj1"
>>> pprint.pprint(dg.outputs.get())
{'T1': <undefined>, 'epi': <undefined>}
```

The same thing with dynamic grabbing of specific files:

```
>>> templates["epi"] = "{subject_id}/func/f{run!s}.nii"
>>> dg = Node(SelectFiles(templates), "selectfiles")
>>> dg.inputs.subject_id = "subj1"
>>> dg.inputs.run = [2, 4]
```

Inputs:

```
[Mandatory]

[Optional]
base_directory: (an existing directory name)
    Root path common to templates.
force_lists: (a boolean or a list of items which are a unicode
    string, nipype default value: False)
    Whether to return outputs as a list even when only one file matches
    the template. Either a boolean that applies to all output fields or
    a list of output field names to coerce to a list
ignore_exception: (a boolean, nipype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
raise_on_empty: (a boolean, nipype default value: True)
    Raise an exception if a template pattern matches no files.
sort_filelist: (a boolean, nipype default value: True)
    When matching mutliple files, return them in sorted order.
```

Outputs:

None

XNATSink

[Link to code](#)

Generic datasink module that takes a directory containing a list of nifti files and provides a set of structured output fields.

Inputs:

```
[Mandatory]
config: (a file name)
    mutually_exclusive: server
experiment_id: (a unicode string)
    Set to workflow name
project_id: (a unicode string)
    Project in which to store the outputs
server: (a unicode string)
    mutually_exclusive: config
    requires: user, pwd
subject_id: (a unicode string)
    Set to subject id

[Optional]
_outputs: (a dictionary with keys which are a unicode string and with
    values which are any value, nipype default value: {})
assessor_id: (a unicode string)
    Option to customize ouputs representation in XNAT - assessor level
    will be used with specified id
    mutually_exclusive: reconstruction_id
```

```

cache_dir: (a directory name)
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
pwd: (a string)
reconstruction_id: (a unicode string)
    Option to customize ouputs representation in XNAT - reconstruction
    level will be used with specified id
mutually_exclusive: assessor_id
share: (a boolean, nipy default value: False)
    Option to share the subjects from the original projectinstead of
    creating new ones when possible - the created experiments are then
    shared back to the original project
user: (a unicode string)

```

Outputs:

None

XNATSource

[Link to code](#)

Generic XNATSource module that wraps around the pyxnat module in an intelligent way for neuroimaging tasks to grab files and data from an XNAT server.

Examples

```
>>> from nipy.interfaces.io import XNATSource
```

Pick all files from current directory

```
>>> dg = XNATSource()
>>> dg.inputs.template = '*'
```

```
>>> dg = XNATSource(infields=['project','subject','experiment','assessor','inout
↪'])
>>> dg.inputs.query_template = '/projects/%s/subjects/%s/experiments/%s'
↪                               '/assessors/%s/%s_resources/files'
>>> dg.inputs.project = 'IMAGEN'
>>> dg.inputs.subject = 'IMAGEN_000000001274'
>>> dg.inputs.experiment = '*SessionA*'
>>> dg.inputs.assessor = '*ADNI_MPRAGE_nii'
>>> dg.inputs.inout = 'out'
```

```
>>> dg = XNATSource(infields=['sid'],outfields=['struct','func'])
>>> dg.inputs.query_template = '/projects/IMAGEN/subjects/%s/experiments/
↪*SessionA*'                               '/assessors/*s_nii/out_resources/files'
>>> dg.inputs.query_template_args['struct'] = [['sid','ADNI_MPRAGE']]
>>> dg.inputs.query_template_args['func'] = [['sid','EPI_faces']]
>>> dg.inputs.sid = 'IMAGEN_000000001274'
```

Inputs:

```

[Mandatory]
config: (a file name)
    mutually_exclusive: server

```

```

query_template: (a unicode string)
    Layout used to get files. Relative to base directory if defined
server: (a unicode string)
    mutually_exclusive: config
    requires: user, pwd

[Optional]
cache_dir: (a directory name)
    Cache directory
ignore_exception: (a boolean, nipyype default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
pwd: (a string)
query_template_args: (a dictionary with keys which are a unicode
    string and with values which are a list of items which are a list
    of items which are any value, nipyype default value: {'outfiles':
    []})
    Information to plug into template
user: (a unicode string)

```

Outputs:

None

add_traits()

[Link to code](#)

Add traits to a traited class.

All traits are set to Undefined by default

copytree()

[Link to code](#)

Recursively copy a directory tree using `nipyype.utils.filemanip.copyfile()`

This is not a thread-safe routine. However, in the case of creating new directories, it checks to see if a particular directory has already been created by another process.

push_file()

[Link to code](#)

quote_id()

[Link to code](#)

unquote_id()

[Link to code](#)

MeshFix

[Link to code](#)

Wraps command **meshfix**

MeshFix v1.2-alpha - by Marco Attene, Mirko Windhoff, Axel Thielscher.

See also:

<http://jmeshlib.sourceforge.net> Sourceforge page

<http://simnibs.de/installation/meshfixandgetfem> Ubuntu installation instructions

If MeshFix is used for research purposes, please cite the following paper: M. Attene - A lightweight approach to repairing digitized polygon meshes. The Visual Computer, 2010. (c) Springer.

Accepted input formats are OFF, PLY and STL. Other formats (like .msh for gmsh) are supported only partially.

Example

```

>>> import nipy.interfaces.meshfix as mf
>>> fix = mf.MeshFix()
>>> fix.inputs.in_file1 = 'lh-pial.stl'
>>> fix.inputs.in_file2 = 'rh-pial.stl'
>>> fix.run()
>>> fix.cmdline
'meshfix lh-pial.stl rh-pial.stl -o lh-pial_fixed.off'

```

Inputs:

```

[Mandatory]
in_file1: (an existing file name)
         flag: %s, position: 1

[Optional]
args: (a unicode string)
      Additional parameters to the command
      flag: %s
cut_inner: (an integer (int or long))
           Remove triangles of 1st that are inside of the 2nd shell. Dilate 2nd
           by N; Fill holes and keep only 1st afterwards.
           flag: --cut-inner %d

```

```
cut_outer: (an integer (int or long))
    Remove triangles of 1st that are outside of the 2nd shell.
    flag: --cut-outer %d
decouple_inin: (an integer (int or long))
    Treat 1st file as inner, 2nd file as outer component. Resolve
    overlaps by moving inners triangles inwards. Constrain the min
    distance between the components > d.
    flag: --decouple-inin %d
decouple_outin: (an integer (int or long))
    Treat 1st file as outer, 2nd file as inner component. Resolve
    overlaps by moving outers triangles inwards. Constrain the min
    distance between the components > d.
    flag: --decouple-outin %d
decouple_outout: (an integer (int or long))
    Treat 1st file as outer, 2nd file as inner component. Resolve
    overlaps by moving outers triangles outwards. Constrain the min
    distance between the components > d.
    flag: --decouple-outout %d
dilation: (an integer (int or long))
    Dilate the surface by d. d < 0 means shrinking.
    flag: --dilate %d
dont_clean: (a boolean)
    Don't Clean
    flag: --no-clean
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipyne default value: {})
    Environment variables
epsilon_angle: (0.0 <= a floating point number <= 2.0)
    Epsilon angle in degrees (must be between 0 and 2)
    flag: -a %f
finetuning_distance: (a float)
    Used to fine-tune the minimal distance between surfaces. A minimal
    distance d is ensured, and reached in n substeps. When using the
    surfaces for subsequent volume meshing by gmsh, this step prevent
    too flat tetrahedra2)
    flag: %f
    requires: finetuning_substeps
finetuning_inwards: (a boolean)
    flag: --fineTuneIn
    requires: finetuning_distance, finetuning_substeps
finetuning_outwards: (a boolean)
    Similar to finetuning_inwards, but ensures minimal distance in the
    other direction
    flag: --fineTuneIn
    mutually_exclusive: finetuning_inwards
    requires: finetuning_distance, finetuning_substeps
finetuning_substeps: (an integer (int or long))
    Used to fine-tune the minimal distance between surfaces. A minimal
    distance d is ensured, and reached in n substeps. When using the
    surfaces for subsequent volume meshing by gmsh, this step prevent
    too flat tetrahedra2)
    flag: %d
    requires: finetuning_distance
ignore_exception: (a boolean, nipyne default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
in_file2: (an existing file name)
```



```

    flag: %s, position: 2
join_closest_components: (a boolean)
    Join the closest pair of components.
    flag: -jc
    mutually_exclusive: join_closest_components
join_overlapping_largest_components: (a boolean)
    Join 2 biggest components if they overlap, remove the rest.
    flag: -j
    mutually_exclusive: join_closest_components
laplacian_smoothing_steps: (an integer (int or long))
    The number of laplacian smoothing steps to apply
    flag: --smooth %d
number_of_biggest_shells: (an integer (int or long))
    Only the N biggest shells are kept
    flag: --shells %d
out_filename: (a file name)
    The output filename for the fixed mesh file
    flag: -o %s
output_type: ('stl' or 'msh' or 'wrl' or 'vrml' or 'fs' or 'off',
    nipyte default value: off)
    The output type to save the file as.
quiet_mode: (a boolean)
    Quiet mode, don't write much to stdout.
    flag: -q
remove_handles: (a boolean)
    Remove handles
    flag: --remove-handles
save_as_freesurfer_mesh: (a boolean)
    Result is saved in freesurfer mesh format
    flag: --fsmesh
    mutually_exclusive: save_as_vrml, save_as_stl
save_as_stl: (a boolean)
    Result is saved in stereolithographic format (.stl)
    flag: --stl
    mutually_exclusive: save_as_vrml, save_as_freesurfer_mesh
save_as_vrml: (a boolean)
    Result is saved in VRML1.0 format (.wrl)
    flag: --wrl
    mutually_exclusive: save_as_stl, save_as_freesurfer_mesh
set_intersections_to_one: (a boolean)
    If the mesh contains intersections, return value = 1.If saved in
    gmsh format, intersections will be highlighted.
    flag: --intersect
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored
uniform_remeshing_steps: (an integer (int or long))
    Number of steps for uniform remeshing of the whole mesh
    flag: -u %d
    requires: uniform_remeshing_vertices
uniform_remeshing_vertices: (an integer (int or long))
    Constrains the number of vertices.Must be used with
    uniform_remeshing_steps
    flag: --vertices %d
    requires: uniform_remeshing_steps
x_shift: (an integer (int or long))
    Shifts the coordinates of the vertices when saving. Output must be

```

```
in FreeSurfer format
flag: --smooth %d
```

Outputs:

```
mesh_file: (an existing file name)
    The output mesh file
```

SignalExtraction

[Link to code](#)

Extracts signals over tissue classes or brain regions

```
>>> seinterface = SignalExtraction()
>>> seinterface.inputs.in_file = 'functional.nii'
>>> seinterface.inputs.label_files = 'segmentation0.nii.gz'
>>> seinterface.inputs.out_file = 'means.tsv'
>>> segments = ['CSF', 'GrayMatter', 'WhiteMatter']
>>> seinterface.inputs.class_labels = segments
>>> seinterface.inputs.detrrend = True
>>> seinterface.inputs.include_global = True
```

Inputs:

```
[Mandatory]
class_labels: (a list of items which are any value)
    Human-readable labels for each segment in the label file, in order.
    The length of class_labels must be equal to the number of segments
    (background excluded). This list corresponds to the class labels in
    label_file in ascending order
in_file: (an existing file name)
    4-D fMRI nii file
label_files: (a list of items which are an existing file name)
    a 3-D label image, with 0 denoting background, or a list of 3-D
    probability maps (one per label) or the equivalent 4D file.

[Optional]
detrrend: (a boolean, nipy default value: False)
    If True, perform detrending using nilearn.
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
incl_shared_variance: (a boolean, nipy default value: True)
    By default (True), returns simple time series calculated from each
    region independently (e.g., for noise regression). If False, returns
```

```
unique signals for each region, discarding shared variance (e.g.,  
for connectivity. Only has effect with 4D probability maps.  
include_global: (a boolean, nipy default value: False)  
    If True, include an extra column labeled "GlobalSignal", with values  
    calculated from the entire brain (instead of just regions).  
out_file: (a file name, nipy default value: signals.tsv)  
    The name of the file to output to. signals.tsv by default
```

Outputs:

```
out_file: (an existing file name)  
    tsv file containing the computed signals, with as many columns as  
    there are labels and as many rows as there are timepoints in  
    in_file, plus a header row with values from class_labels
```

PETPVC

[Link to code](#)

Wraps command **petpvc**

Use PETPVC for partial volume correction of PET images.

PETPVC is a software from the Nuclear Medicine Department of the UCL University Hospital, London, UK.

Its source code is here: <https://github.com/UCL/PETPVC>

The methods that it implement are explained here: K. Erlandsson, I. Buvat, P. H. Pretorius, B. A. Thomas, and B. F. Hutton, “A review of partial volume correction techniques for emission tomography and their applications in neurology, cardiology and oncology,” Phys. Med. Biol., vol. 57, no. 21, p. R119, 2012.

Its command line help shows this:

```
-i -input < filename > = PET image file
-o -output < filename > = Output file
-m -mask < filename > ] = Mask image file
-p -pvc < keyword > = Desired PVC method
    -x < X >           = The full-width at half maximum in mm along x-axis
    -y < Y >           = The full-width at half maximum in mm along y-axis
    -z < Z >           = The full-width at half maximum in mm along z-axis
-d -debug ] = Prints debug information
-n -iter [ Val ]
    = Number of iterations With: Val (Default = 10)
-k [ Val ]
    = Number of deconvolution iterations With: Val (Default = 10)
-a -alpha [ aval ]
    = Alpha value With: aval (Default = 1.5)
-s -stop [ stopval ]
    = Stopping criterion With: stopval (Default = 0.01)
```

Technique - keyword

- Geometric transfer matrix - “GTM”
- Labbe approach - “LABBE”
- Richardson-Lucy - “RL”
- Van-Cittert - “VC”

- Region-based voxel-wise correction - “RBV”
- RBV with Labbe - “LABBE+RBV”
- RBV with Van-Cittert - “RBV+VC”
- RBV with Richardson-Lucy - “RBV+RL”
- RBV with Labbe and Van-Cittert - “LABBE+RBV+VC”
- RBV with Labbe and Richardson-Lucy- “LABBE+RBV+RL”
- Multi-target correction - “MTC”
- MTC with Labbe - “LABBE+MTC”
- MTC with Van-Cittert - “MTC+VC”
- MTC with Richardson-Lucy - “MTC+RL”
- MTC with Labbe and Van-Cittert - “LABBE+MTC+VC”
- MTC with Labbe and Richardson-Lucy- “LABBE+MTC+RL”
- Iterative Yang - “IY”
- Iterative Yang with Van-Cittert - “IY+VC”
- Iterative Yang with Richardson-Lucy - “IY+RL”
- Muller Gartner - “MG”
- Muller Gartner with Van-Cittert - “MG+VC”
- Muller Gartner with Richardson-Lucy - “MG+RL”

Examples

```
>>> from ..testing import example_data
>>> #TODO get data for PETPVC
>>> pvc = PETPVC()
>>> pvc.inputs.in_file = 'pet.nii.gz'
>>> pvc.inputs.mask_file = 'tissues.nii.gz'
>>> pvc.inputs.out_file = 'pet_pvc_rbv.nii.gz'
>>> pvc.inputs.pvc = 'RBV'
>>> pvc.inputs.fwhm_x = 2.0
>>> pvc.inputs.fwhm_y = 2.0
>>> pvc.inputs.fwhm_z = 2.0
>>> outs = pvc.run()
```

Inputs:

```
[Mandatory]
fwhm_x: (a float)
    The full-width at half maximum in mm along x-axis
    flag: -x %.4f
fwhm_y: (a float)
    The full-width at half maximum in mm along y-axis
    flag: -y %.4f
fwhm_z: (a float)
    The full-width at half maximum in mm along z-axis
    flag: -z %.4f
in_file: (an existing file name)
    PET image file
    flag: -i %s
mask_file: (an existing file name)
    Mask image file
    flag: -m %s
pvc: ('GTM' or 'IY' or 'IY+RL' or 'IY+VC' or 'LABBE' or 'LABBE+MTC'
    or 'LABBE+MTC+RL' or 'LABBE+MTC+VC' or 'LABBE+RBV' or
    'LABBE+RBV+RL' or 'LABBE+RBV+VC' or 'MG' or 'MG+RL' or 'MG+VC' or
    'MTC' or 'MTC+RL' or 'MTC+VC' or 'RBV' or 'RBV+RL' or 'RBV+VC' or
    'RL' or 'VC')
    Desired PVC method
```

```

        flag: -p %s

[Optional]
alpha: (a float)
    Alpha value
    flag: -a %.4f
args: (a unicode string)
    Additional parameters to the command
    flag: %s
debug: (a boolean, nipy default value: False)
    Prints debug information
    flag: -d
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a value
    of class 'str', nipy default value: {})
    Environment variables
ignore_exception: (a boolean, nipy default value: False)
    Print an error message instead of throwing an exception in case the
    interface fails to run
n_deconv: (an integer (int or long))
    Number of deconvolution iterations
    flag: -k %d
n_iter: (an integer (int or long))
    Number of iterations
    flag: -n %d
out_file: (a file name)
    Output file
    flag: -o %s
stop_crit: (a float)
    Stopping criterion
    flag: -a %.4f
terminal_output: ('stream' or 'allatonce' or 'file' or 'none')
    Control terminal output: `stream` - displays to terminal immediately
    (default), `allatonce` - waits till command is finished to display
    output, `file` - writes output to file, `none` - output is ignored

```

Outputs:

```

out_file: (a file name)
    Output file

```

References:: None

configure_input_data()

[Link to code](#)

Configure the input data for vtk pipeline object obj. Copied from latest version of mayavi

vtk_old()

[Link to code](#)

Checks if VTK uses the old-style pipeline (VTK<6.0)

vtk_output()

[Link to code](#)

Configure the input data for vtk pipeline object obj.

Bibliography

- [R1] A. Martelli, D. Hudgeon, "Collecting a Bunch of Named Items", Python Cookbook, 2nd Ed, Chapter 4.18, 2005.
- [Jones10] Jones DK, The signal intensity must be modulated by the determinant of the Jacobian when correcting for eddy currents in diffusion MRI, Proc. ISMRM 18th Annual Meeting, (2010).
- [Rohde04] Rohde et al., Comprehensive Approach for Correction of Motion and Distortion in Diffusion-Weighted MRI, MRM 51:103-114 (2004).
- [Leemans09] Leemans A, and Jones DK, The B-matrix must be rotated when correcting for subject motion in DTI data, Magn Reson Med. 61(6):1336-49. 2009. doi: 10.1002/mrm.21890.
- [Yendiki13] Yendiki A et al., Spurious group differences due to head motion in a diffusion MRI study. Neuroimage. 21(88C):79-90. 2013. doi: 10.1016/j.neuroimage.2013.11.027
- [Jeurissen2014] Jeurissen B. et al., Multi-tissue constrained spherical deconvolution for improved analysis of multi-shell diffusion MRI data. NeuroImage (2014). doi: 10.1016/j.neuroimage.2014.07.061
- [Jezzard95] Jezzard P, and Balaban RS, Correction for geometric distortion in echo planar images from B0 field variations, MRM 34(1):65-73. (1995). doi: 10.1002/mrm.1910340111.
- [Jenkinson03] Jenkinson M., Fast, automated, N-dimensional phase-unwrapping algorithm, MRM 49(1):193-197, 2003, doi: 10.1002/mrm.10354.
- [Andersson2003] Andersson JL et al., How to correct susceptibility distortions in spin-echo echo-planar images: application to diffusion tensor imaging. Neuroimage. 2003 Oct;20(2):870-88. doi: 10.1016/S1053-8119(03)00336-7
- [Cordes2000] Cordes D et al., Geometric distortion correction in EPI using two images with orthogonal phase-encoding directions, in Proc. ISMRM (8), p.1712, Denver, US, 2000.
- [Chiou2000] Chiou JY, and Nalcioğlu O, A simple method to correct off-resonance related distortion in echo planar imaging, in Proc. ISMRM (8), p.1712, Denver, US, 2000.
- [Power2012] Power et al., Spurious but systematic correlations in functional connectivity MRI networks arise from subject motion, NeuroImage 59(3), 2012. doi:10.1016/j.neuroimage.2011.10.018.
- [Nichols2013] Nichols T, Notes on creating a standardized version of DVARS, 2013.
- [Tustison2010] N. Tustison et al., N4ITK: Improved N3 Bias Correction, IEEE Transactions on Medical Imaging, 29(6):1310-1320, June 2010.
- [Coupe2008] Coupe P et al., An Optimized Blockwise Non Local Means Denoising Filter for 3D Magnetic Resonance Images, IEEE Transactions on Medical Imaging, 27(4):425-441, 2008.

- [Tournier2007] Tournier, J.D., et al. NeuroImage 2007. Robust determination of the fibre orientation distribution in diffusion MRI: Non-negativity constrained super-resolved spherical deconvolution
- [Chang2005] Chang, LC, Jones, DK and Pierpaoli, C. RESTORE: robust estimation of tensors by outlier rejection. MRM, 53:1088-95, (2005).
- [Garyfallidis12] Garyfallidis E., “Towards an accurate brain tractography”, PhD thesis, University of Cambridge, 2012
- [FACT] Mori, S.; Crain, B. J.; Chacko, V. P. & van Zijl, P. C. M. Three-dimensional tracking of axonal projections in the brain by magnetic resonance imaging. Annals of Neurology, 1999, 45, 265-269
- [iFOD1] Tournier, J.-D.; Calamante, F. & Connelly, A. MRtrix: Diffusion tractography in crossing fiber regions. Int. J. Imaging Syst. Technol., 2012, 22, 53-66
- [iFOD2] Tournier, J.-D.; Calamante, F. & Connelly, A. Improved probabilistic streamlines tractography by 2nd order integration over fibre orientation distributions. Proceedings of the International Society for Magnetic Resonance in Medicine, 2010, 1670
- [Nulldist] Morris, D. M.; Embleton, K. V. & Parker, G. J. Probabilistic fibre tracking: Differentiation of connections from chance events. NeuroImage, 2008, 42, 1329-1339
- [Tensor_Det] Basser, P. J.; Pajevic, S.; Pierpaoli, C.; Duda, J. and Aldroubi, A. In vivo fiber tractography using DT-MRI data. Magnetic Resonance in Medicine, 2000, 44, 625-632
- [Tensor_Prob] Jones, D. Tractography Gone Wild: Probabilistic Fibre Tracking Using the Wild Bootstrap With Diffusion Tensor MRI. IEEE Transactions on Medical Imaging, 2008, 27, 1268-1274
- [Roche2011] Roche A. A four-dimensional registration algorithm with application to joint correction of motion and slice timing in fMRI. IEEE Trans Med Imaging. 2011 Aug;30(8):1546-54. [DOI](#).

n

nipype.algorithms.confounds, 557
nipype.algorithms.icc, 559
nipype.algorithms.misc, 582
nipype.algorithms.modelgen, 590
nipype.caching.memory, 60
nipype.interfaces.base, 62
nipype.interfaces.brainsuite.brainsuite, 746
nipype.interfaces.cmtk.cmtk, 818
nipype.interfaces.cmtk.nbs, 822
nipype.interfaces.cmtk.nx, 824
nipype.interfaces.cmtk.parcellation, 826
nipype.interfaces.dcmstack, 1620
nipype.interfaces.dipy.preprocess, 844
nipype.interfaces.freesurfer.utils, 988
nipype.interfaces.io, 1639
nipype.interfaces.matlab, 144
nipype.interfaces.minc.base, 1137
nipype.interfaces.minc.testdata, 1186
nipype.interfaces.mrtrix.convert, 1217
nipype.interfaces.mrtrix.tensors, 1238
nipype.interfaces.niftyreg.base, 1289
nipype.interfaces.vtkbase, 1651
nipype.pipeline.engine.base, 155
nipype.pipeline.engine.nodes, 156
nipype.pipeline.engine.utils, 161
nipype.pipeline.engine.workflows, 164
nipype.workflows.data, 203
nipype.workflows.dmri.camino.connectivity_mapping, 205
nipype.workflows.dmri.camino.diffusion, 206
nipype.workflows.dmri.camino.group_connectivity, 207
nipype.workflows.dmri.connectivity.group_connectivity, 208
nipype.workflows.dmri.connectivity.nx, 211
nipype.workflows.dmri.dipy.denoise, 213
nipype.workflows.dmri.fsl.artifacts, 214
nipype.workflows.dmri.fsl.dti, 231
nipype.workflows.dmri.fsl.epi, 235
nipype.workflows.dmri.fsl.tbss, 247
nipype.workflows.dmri.fsl.utils, 259
nipype.workflows.dmri.mrtrix.connectivity_mapping, 265
nipype.workflows.dmri.mrtrix.diffusion, 266
nipype.workflows.dmri.mrtrix.group_connectivity, 268
nipype.workflows.fmri.fsl.estimate, 269
nipype.workflows.fmri.fsl.preprocess, 273
nipype.workflows.fmri.spm.preprocess, 282
nipype.workflows.misc.utils, 289
nipype.workflows.rsfmri.fsl.resting, 291
nipype.workflows.smri.ants.ANTSBuildTemplate, 297
nipype.workflows.smri.ants.antsRegistrationBuildTemplate, 299
nipype.workflows.smri.freesurfer.autorecon1, 300
nipype.workflows.smri.freesurfer.autorecon2, 300
nipype.workflows.smri.freesurfer.bem, 300
nipype.workflows.smri.freesurfer.recon, 302
nipype.workflows.smri.freesurfer.utils, 304
nipype.workflows.smri.niftyreg.groupwise, 310

Symbols

- `__init__()` (nipyype.caching.memory.Memory method), 60
 - `__init__()` (nipyype.caching.memory.PipeFunc method), 8, 61
 - `__init__()` (nipyype.interfaces.base.BaseInterface method), 63
 - `__init__()` (nipyype.interfaces.base.BaseInterfaceInputSpec method), 65
 - `__init__()` (nipyype.interfaces.base.BaseTraitedSpec method), 74
 - `__init__()` (nipyype.interfaces.base.Bunch method), 81
 - `__init__()` (nipyype.interfaces.base.CommandLine method), 82
 - `__init__()` (nipyype.interfaces.base.CommandLineInputSpec method), 85
 - `__init__()` (nipyype.interfaces.base.DynamicTraitedSpec method), 94
 - `__init__()` (nipyype.interfaces.base.InputMultiPath method), 101
 - `__init__()` (nipyype.interfaces.base.Interface method), 104
 - `__init__()` (nipyype.interfaces.base.MpiCommandLine method), 105
 - `__init__()` (nipyype.interfaces.base.MpiCommandLineInputSpec method), 108
 - `__init__()` (nipyype.interfaces.base.MultiPath method), 115
 - `__init__()` (nipyype.interfaces.base.NipyypeInterfaceError method), 117
 - `__init__()` (nipyype.interfaces.base.OutputMultiPath method), 119
 - `__init__()` (nipyype.interfaces.base.SEMLikeCommandLine method), 121
 - `__init__()` (nipyype.interfaces.base.StdOutCommandLine method), 122
 - `__init__()` (nipyype.interfaces.base.StdOutCommandLineInputSpec method), 125
 - `__init__()` (nipyype.interfaces.base.Str method), 132
 - `__init__()` (nipyype.interfaces.base.Stream method), 134
 - `__init__()` (nipyype.interfaces.base.TraitedSpec method), 137
 - `__init__()` (nipyype.interfaces.matlab.MatlabCommand method), 145
 - `__init__()` (nipyype.interfaces.matlab.MatlabInputSpec method), 148
 - `__init__()` (nipyype.pipeline.engine.base.EngineBase method), 155
 - `__init__()` (nipyype.pipeline.engine.nodes.JoinNode method), 157
 - `__init__()` (nipyype.pipeline.engine.nodes.MapNode method), 158
 - `__init__()` (nipyype.pipeline.engine.nodes.Node method), 160
 - `__init__()` (nipyype.pipeline.engine.workflows.Workflow method), 164
 - `__init__()` (nipyype.sphinxext.plot_workflow.GraphError method), 169
 - `__init__()` (nipyype.sphinxext.plot_workflow.ImageFile method), 169
- ## A
- ACompCor, 551
 - AGPCTransform, 1539
 - `add_class_trait()` (nipyype.interfaces.base.BaseInterfaceInputSpec method), 65
 - `add_class_trait()` (nipyype.interfaces.base.BaseTraitedSpec method), 74
 - `add_class_trait()` (nipyype.interfaces.base.CommandLineInputSpec method), 85
 - `add_class_trait()` (nipyype.interfaces.base.DynamicTraitedSpec method), 94
 - `add_class_trait()` (nipyype.interfaces.base.MpiCommandLineInputSpec method), 108
 - `add_class_trait()` (nipyype.interfaces.base.StdOutCommandLineInputSpec method), 125
 - `add_class_trait()` (nipyype.interfaces.base.TraitedSpec method), 137
 - `add_class_trait()` (nipyype.interfaces.matlab.MatlabInputSpec method), 148

[add_nodes\(\) \(nipype.pipeline.engine.workflows.Workflow method\), 164](#)
[add_trait\(\) \(nipype.interfaces.base.BaseInterfaceInputSpec method\), 65](#)
[add_trait\(\) \(nipype.interfaces.base.BaseTraitedSpec method\), 74](#)
[add_trait\(\) \(nipype.interfaces.base.CommandLineInputSpec method\), 85](#)
[add_trait\(\) \(nipype.interfaces.base.DynamicTraitedSpec method\), 94](#)
[add_trait\(\) \(nipype.interfaces.base.MpiCommandLineInputSpec method\), 108](#)
[add_trait\(\) \(nipype.interfaces.base.StdOutCommandLineInputSpec method\), 125](#)
[add_trait\(\) \(nipype.interfaces.base.TraitedSpec method\), 137](#)
[add_trait\(\) \(nipype.interfaces.matlab.MatlabInputSpec method\), 148](#)
[add_trait_category\(\) \(nipype.interfaces.base.BaseInterfaceInputSpec method\), 66](#)
[add_trait_category\(\) \(nipype.interfaces.base.BaseTraitedSpec method\), 74](#)
[add_trait_category\(\) \(nipype.interfaces.base.CommandLineInputSpec method\), 85](#)
[add_trait_category\(\) \(nipype.interfaces.base.DynamicTraitedSpec method\), 94](#)
[add_trait_category\(\) \(nipype.interfaces.base.MpiCommandLineInputSpec method\), 108](#)
[add_trait_category\(\) \(nipype.interfaces.base.StdOutCommandLineInputSpec method\), 125](#)
[add_trait_category\(\) \(nipype.interfaces.base.TraitedSpec method\), 137](#)
[add_trait_category\(\) \(nipype.interfaces.matlab.MatlabInputSpec method\), 148](#)
[add_trait_listener\(\) \(nipype.interfaces.base.BaseInterfaceInputSpec method\), 66](#)
[add_trait_listener\(\) \(nipype.interfaces.base.BaseTraitedSpec method\), 74](#)
[add_trait_listener\(\) \(nipype.interfaces.base.CommandLineInputSpec method\), 85](#)
[add_trait_listener\(\) \(nipype.interfaces.base.DynamicTraitedSpec method\), 94](#)
[add_trait_listener\(\) \(nipype.interfaces.base.MpiCommandLineInputSpec method\), 108](#)
[add_trait_listener\(\) \(nipype.interfaces.base.StdOutCommandLineInputSpec method\), 125](#)
[add_trait_listener\(\) \(nipype.interfaces.base.TraitedSpec method\), 137](#)
[add_trait_listener\(\) \(nipype.interfaces.matlab.MatlabInputSpec method\), 148](#)
[AddCSVColumn, 571](#)
[AddCSVRow, 571](#)
[AddNoise, 572](#)
[AddScalarVolumes, 1489](#)
[AddXFormToHeader, 938](#)
[AffineInitializer, 707](#)
[AffineRegistration, 1515](#)
[AFNItoNIFTI, 641](#)
[aggregate_outputs\(\) \(nipype.interfaces.base.BaseInterface method\), 63](#)
[aggregate_outputs\(\) \(nipype.interfaces.base.CommandLine method\), 82](#)
[aggregate_outputs\(\) \(nipype.interfaces.base.Interface method\), 104](#)
[aggregate_outputs\(\) \(nipype.interfaces.base.MpiCommandLine method\), 105](#)
[aggregate_outputs\(\) \(nipype.interfaces.base.SEMLikeCommandLine method\), 121](#)
[aggregate_outputs\(\) \(nipype.interfaces.base.StdOutCommandLine method\), 122](#)
[aggregate_outputs\(\) \(nipype.interfaces.matlab.MatlabCommand method\), 145](#)
[all_trait_names\(\) \(nipype.interfaces.base.BaseInterfaceInputSpec method\), 66](#)
[all_trait_names\(\) \(nipype.interfaces.base.BaseTraitedSpec method\), 74](#)
[all_trait_names\(\) \(nipype.interfaces.base.CommandLineInputSpec method\), 85](#)
[all_trait_names\(\) \(nipype.interfaces.base.DynamicTraitedSpec method\), 94](#)
[all_trait_names\(\) \(nipype.interfaces.base.MpiCommandLineInputSpec method\), 108](#)
[all_trait_names\(\) \(nipype.interfaces.base.StdOutCommandLineInputSpec method\), 125](#)
[all_trait_names\(\) \(nipype.interfaces.base.TraitedSpec method\), 137](#)
[all_trait_names\(\) \(nipype.interfaces.matlab.MatlabInputSpec method\), 148](#)
[AllSpec, 597](#)
[always_run \(nipype.interfaces.base.BaseInterface attribute\), 63](#)
[always_run \(nipype.interfaces.base.CommandLine attribute\), 82](#)
[always_run \(nipype.interfaces.base.Interface attribute\), 104](#)
[always_run \(nipype.interfaces.base.MpiCommandLine attribute\), 105](#)
[always_run \(nipype.interfaces.base.SEMLikeCommandLine attribute\), 121](#)
[always_run \(nipype.interfaces.base.StdOutCommandLine attribute\), 122](#)
[always_run \(nipype.interfaces.matlab.MatlabCommand attribute\), 145](#)
[Analyze2nii, 1593](#)
[AnalyzeHeader, 753](#)
[AnalyzeWarp, 853](#)
[antsBrainExtraction, 702](#)
[antsCorticalThickness, 704](#)

- antsIntroduction, 666
- AntsJointFusion, 684
- Aparc2Aseg, 939
- Apas2Aseg, 941
- APMQball, 841
- ApplyDeformations, 1575
- ApplyInverseDeformation, 1594
- ApplyMask, 941, 1030
- ApplyTOPUP, 1011
- ApplyTransform, 1595
- ApplyTransforms, 678
- ApplyTransformsToPoints, 680
- ApplyVolTransform, 893
- ApplyWarp, 854, 1074
- ApplyXFM, 1075
- AR1Image, 1029
- args (nipy.interfaces.base.NipyInterfaceError attribute), 117
- args (nipy.sphinxext.plot_workflow.GraphError attribute), 169
- ArtifactDetect, 593
- as_ctrait() (nipy.interfaces.base.InputMultiPath method), 102
- as_ctrait() (nipy.interfaces.base.MultiPath method), 115
- as_ctrait() (nipy.interfaces.base.OutputMultiPath method), 119
- as_ctrait() (nipy.interfaces.base.Str method), 132
- AssertEqual, 1601
- Atropos, 687
- Autobox, 642
- Automask, 602
- AutoTcorrelate, 601
- Average, 1137
- AverageAffineTransform, 708
- AverageImages, 709
- AverageNetworks, 822
- AvScale, 1102
- B**
- B0Calc, 1072
- Bandpass, 603
- base_trait() (nipy.interfaces.base.BaseInterfaceInputSpec method), 66
- base_trait() (nipy.interfaces.base.BaseTraitedSpec method), 74
- base_trait() (nipy.interfaces.base.CommandLineInputSpec method), 85
- base_trait() (nipy.interfaces.base.DynamicTraitedSpec method), 94
- base_trait() (nipy.interfaces.base.MpiCommandLineInputSpec method), 108
- base_trait() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 125
- base_trait() (nipy.interfaces.base.TraitedSpec method), 137
- base_trait() (nipy.interfaces.matlab.MatlabInputSpec method), 148
- BaseInterface (class in nipy.interfaces.base), 62
- BaseInterfaceInputSpec (class in nipy.interfaces.base), 63
- BaseTraitedSpec (class in nipy.interfaces.base), 72
- BBox, 1140
- BBRegister, 895
- BDP, 717
- Beast, 1142
- BEDPOSTX5, 991
- BestLinReg, 1143
- BET, 1079
- Bfc, 725
- BigAverage, 1144
- Binarize, 862
- BinaryMaskEditorBasedOnLandmarks, 1449
- BinaryMaths, 1030, 1314
- BinaryMathsInteger, 1316
- BinaryStats, 1325
- Blob, 1146
- Blur, 1147
- BlurInMask, 605
- BlurToFWHM, 606
- BrainExtraction, 689
- BrainMask, 1266
- BRAINSABC, 1437
- BRAINSAlignMSP, 1452
- BRAINSClipInferior, 1453
- BRAINSConstellationDetector, 1440
- BRAINSConstellationModeler, 1454
- BRAINSCreateLabelMapFromProbabilityMaps, 1444
- BRAINSCut, 1445
- BRAINSDemonWarp, 1429, 1540
- BRAINSEyeDetector, 1456
- BRAINSFit, 1418, 1530
- BRAINSInitializedControlPoints, 1456
- BRAINSLandmarkInitializer, 1457
- BRAINSLinearModelerEPCA, 1458
- BRAINSLmkTransform, 1459
- BRAINSMultiSTAPLE, 1447
- BRAINSMush, 1460
- BRAINSPosteriorToContinuousClass, 1341
- BRAINSResample, 1426, 1537
- BRAINSResize, 1428
- BRAINSROIAuto, 1448, 1549
- BRAINSSnapShotWriter, 1462
- BRAINSTalairach, 1342
- BRAINSTalairachMask, 1343
- BRAINSTransformConvert, 1463
- BRAINSTransformFromFiducials, 1432
- BRAINSTrimForegroundInDirection, 1464

BrickStat, 643
 Bru2, 1609
 Bse, 728
 BSplineDeformableRegistration, 1517
 BSplineToDeformationField, 1511
 BuildConnectome, 1253
 buildtemplateparallel, 668
 Bunch (class in nipy.interfaces.base), 81

C

C3dAffineTool, 1611
 cache() (nipy.caching.Memory method), 7
 cache() (nipy.caching.memory.Memory method), 60
 CALabel, 897
 Calc, 644, 1149
 CalcCoregAffine, 1595
 CalcTopNCC, 1309
 CalculateNormalizedMoments, 573
 Camino2Trackvis, 811
 can_resume (nipy.interfaces.base.BaseInterface attribute), 63
 can_resume (nipy.interfaces.base.CommandLine attribute), 82
 can_resume (nipy.interfaces.base.Interface attribute), 104
 can_resume (nipy.interfaces.base.MpiCommandLine attribute), 105
 can_resume (nipy.interfaces.base.SEMLikeCommandLine attribute), 121
 can_resume (nipy.interfaces.base.StdOutCommandLine attribute), 122
 can_resume (nipy.interfaces.matlab.MatlabCommand attribute), 145
 CannyEdge, 1402
 CannySegmentationLevelSetImageFilter, 1403
 CANormalize, 898
 CARegister, 900
 CastScalarVolume, 1490
 Cerebro, 729
 CFFConverter, 819
 ChangeDataType, 1032
 CheckerBoardFilter, 1494
 CheckTalairachAlignment, 943
 class_default_traits_view()
 (nipy.interfaces.base.BaseInterfaceInputSpec method), 66
 class_default_traits_view()
 (nipy.interfaces.base.BaseTraitedSpec method), 74
 class_default_traits_view()
 (nipy.interfaces.base.CommandLineInputSpec method), 86
 class_default_traits_view()
 (nipy.interfaces.base.DynamicTraitedSpec method), 94
 class_default_traits_view()
 (nipy.interfaces.base.MpiCommandLineInputSpec method), 108
 class_default_traits_view()
 (nipy.interfaces.base.StdOutCommandLineInputSpec method), 125
 class_default_traits_view()
 (nipy.interfaces.base.TraitedSpec method), 137
 class_default_traits_view()
 (nipy.interfaces.matlab.MatlabInputSpec method), 148
 class_editable_traits() (nipy.interfaces.base.BaseInterfaceInputSpec method), 66
 class_editable_traits() (nipy.interfaces.base.BaseTraitedSpec method), 74
 class_editable_traits() (nipy.interfaces.base.CommandLineInputSpec method), 86
 class_editable_traits() (nipy.interfaces.base.DynamicTraitedSpec method), 94
 class_editable_traits() (nipy.interfaces.base.MpiCommandLineInputSpec method), 108
 class_editable_traits() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 125
 class_editable_traits() (nipy.interfaces.base.TraitedSpec method), 137
 class_editable_traits() (nipy.interfaces.matlab.MatlabInputSpec method), 148
 class_trait_names() (nipy.interfaces.base.BaseInterfaceInputSpec method), 66
 class_trait_names() (nipy.interfaces.base.BaseTraitedSpec method), 75
 class_trait_names() (nipy.interfaces.base.CommandLineInputSpec method), 86
 class_trait_names() (nipy.interfaces.base.DynamicTraitedSpec method), 94
 class_trait_names() (nipy.interfaces.base.MpiCommandLineInputSpec method), 108
 class_trait_names() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 125
 class_trait_names() (nipy.interfaces.base.TraitedSpec method), 137
 class_trait_names() (nipy.interfaces.matlab.MatlabInputSpec method), 148
 class_trait_view() (nipy.interfaces.base.BaseInterfaceInputSpec method), 66
 class_trait_view() (nipy.interfaces.base.BaseTraitedSpec method), 75
 class_trait_view() (nipy.interfaces.base.CommandLineInputSpec method), 86
 class_trait_view() (nipy.interfaces.base.DynamicTraitedSpec method), 94
 class_trait_view() (nipy.interfaces.base.MpiCommandLineInputSpec method), 108
 class_trait_view() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 125
 class_trait_view() (nipy.interfaces.base.TraitedSpec method), 137
 class_trait_view() (nipy.interfaces.matlab.MatlabInputSpec method), 148

1661

[cmd](#) (nipyype.interfaces.base.CommandLine attribute), [82](#)
[cmd](#) (nipyype.interfaces.base.MpiCommandLine attribute), [105](#)
[cmd](#) (nipyype.interfaces.base.SEMLikeCommandLine attribute), [121](#)
[cmd](#) (nipyype.interfaces.base.StdOutCommandLine attribute), [122](#)
[cmd](#) (nipyype.interfaces.matlab.MatlabCommand attribute), [145](#)
[cmdline](#) (nipyype.interfaces.base.CommandLine attribute), [82](#)
[cmdline](#) (nipyype.interfaces.base.MpiCommandLine attribute), [105](#)
[cmdline](#) (nipyype.interfaces.base.SEMLikeCommandLine attribute), [121](#)
[cmdline](#) (nipyype.interfaces.base.StdOutCommandLine attribute), [122](#)
[cmdline](#) (nipyype.interfaces.matlab.MatlabCommand attribute), [145](#)
[CoherenceAnalyzer](#), [1339](#)
[CommandLine](#) (class in nipyype.interfaces.base), [81](#)
[CommandLineInputSpec](#) (class in nipyype.interfaces.base), [83](#)
[compareTractInclusion](#), [1359](#)
[CompCor](#), [552](#)
[Complex](#), [1103](#)
[ComputeDVARs](#), [553](#)
[ComputeEigensystem](#), [765](#)
[ComputeFractionalAnisotropy](#), [766](#)
[ComputeMask](#), [1333](#)
[ComputeMeanDiffusivity](#), [767](#)
[ComputeMeshWarp](#), [561](#)
[ComputeTDI](#), [1267](#)
[ComputeTensorTrace](#), [768](#)
[Concatenate](#), [864](#)
[ConcatenateLTA](#), [901](#)
[configure_traits\(\)](#) (nipyype.interfaces.base.BaseInterfaceInputSpec method), [66](#)
[configure_traits\(\)](#) (nipyype.interfaces.base.BaseTraitedSpec method), [75](#)
[configure_traits\(\)](#) (nipyype.interfaces.base.CommandLineInputSpec method), [86](#)
[configure_traits\(\)](#) (nipyype.interfaces.base.DynamicTraitedSpec method), [95](#)
[configure_traits\(\)](#) (nipyype.interfaces.base.MpiCommandLineInputSpec method), [109](#)
[configure_traits\(\)](#) (nipyype.interfaces.base.StdOutCommandLineInputSpec method), [126](#)
[configure_traits\(\)](#) (nipyype.interfaces.base.TraitdSpec method), [138](#)
[configure_traits\(\)](#) (nipyype.interfaces.matlab.MatlabInputSpec method), [149](#)
[Conmat](#), [751](#)
[connect\(\)](#) (nipyype.pipeline.engine.workflows.Workflow method), [165](#)
[ConstrainedSphericalDeconvolution](#), [1230](#)
[contains_doctest\(\)](#) (in module nipyype.sphinxext.plot_workflow), [169](#)
[Contrast](#), [944](#)
[ContrastMgr](#), [1051](#)
[Convert](#), [1152](#)
[ConvertScalarImageToRGB](#), [712](#)
[ConvertWarp](#), [1105](#)
[ConvertXFM](#), [1107](#)
[Copy](#), [646](#), [1153](#)
[copy_traits\(\)](#) (nipyype.interfaces.base.BaseInterfaceInputSpec method), [67](#)
[copy_traits\(\)](#) (nipyype.interfaces.base.BaseTraitedSpec method), [76](#)
[copy_traits\(\)](#) (nipyype.interfaces.base.CommandLineInputSpec method), [87](#)
[copy_traits\(\)](#) (nipyype.interfaces.base.DynamicTraitedSpec method), [95](#)
[copy_traits\(\)](#) (nipyype.interfaces.base.MpiCommandLineInputSpec method), [109](#)
[copy_traits\(\)](#) (nipyype.interfaces.base.StdOutCommandLineInputSpec method), [127](#)
[copy_traits\(\)](#) (nipyype.interfaces.base.TraitdSpec method), [138](#)
[copy_traits\(\)](#) (nipyype.interfaces.matlab.MatlabInputSpec method), [149](#)
[copyable_trait_names\(\)](#) (nipyype.interfaces.base.BaseInterfaceInputSpec method), [67](#)
[copyable_trait_names\(\)](#) (nipyype.interfaces.base.BaseTraitedSpec method), [76](#)
[copyable_trait_names\(\)](#) (nipyype.interfaces.base.CommandLineInputSpec method), [87](#)
[copyable_trait_names\(\)](#) (nipyype.interfaces.base.DynamicTraitedSpec method), [96](#)
[copyable_trait_names\(\)](#) (nipyype.interfaces.base.MpiCommandLineInputSpec method), [110](#)
[copyable_trait_names\(\)](#) (nipyype.interfaces.base.StdOutCommandLineInputSpec method), [127](#)
[copyable_trait_names\(\)](#) (nipyype.interfaces.base.TraitdSpec method), [139](#)
[copyable_trait_names\(\)](#) (nipyype.interfaces.matlab.MatlabInputSpec method), [150](#)
[CopyGeom](#), [1108](#)
[CopySpec](#), [1617](#)
[Coregister](#), [1576](#)
[CorticalInputSpec](#), [1617](#)
[CorticalThickness](#), [691](#)
[count_iterables\(\)](#) (in module nipyype.pipeline.engine.utils), [161](#)
[create_editor\(\)](#) (nipyype.interfaces.base.InputMultiPath method), [102](#)
[create_editor\(\)](#) (nipyype.interfaces.base.MultiPath method), [115](#)

- [create_editor\(\) \(nipype.interfaces.base.OutputMultiPath method\), 119](#)
[create_editor\(\) \(nipype.interfaces.base.Str method\), 132](#)
[CreateJacobianDeterminantImage, 710](#)
[CreateMatrix, 815](#)
[CreateNifti, 573](#)
[CreateNodes, 816](#)
[CreateTiledMosaic, 713](#)
[CreateWarped, 1577](#)
[CSD, 844](#)
[CSVReader, 1605](#)
[Curvature, 945](#)
[CurvatureAnisotropicDiffusion, 1495](#)
[CurvatureStats, 946](#)
- ## D
- [DARTEL, 1578](#)
[DARTELNorm2MNI, 1579](#)
[DataFinder, 1625](#)
[DataGrabber, 1626](#)
[DataSink, 1627](#)
[Dcm2nii, 1613](#)
[Dcm2niix, 1615](#)
[DcmStack, 1617](#)
[default_traits_view\(\) \(nipype.interfaces.base.BaseInterfaceInputSpec method\), 67](#)
[default_traits_view\(\) \(nipype.interfaces.base.BaseTraitedSpec method\), 76](#)
[default_traits_view\(\) \(nipype.interfaces.base.CommandLineInputSpec method\), 87](#)
[default_traits_view\(\) \(nipype.interfaces.base.DynamicTraitedSpec method\), 96](#)
[default_traits_view\(\) \(nipype.interfaces.base.MpiCommandLineInputSpec method\), 110](#)
[default_traits_view\(\) \(nipype.interfaces.base.StdOutCommandLineInputSpec method\), 127](#)
[default_traits_view\(\) \(nipype.interfaces.base.TraitSpec method\), 139](#)
[default_traits_view\(\) \(nipype.interfaces.matlab.MatlabInputSpec method\), 150](#)
[default_value \(nipype.interfaces.base.InputMultiPath attribute\), 102](#)
[default_value \(nipype.interfaces.base.MultiPath attribute\), 115](#)
[default_value \(nipype.interfaces.base.OutputMultiPath attribute\), 119](#)
[default_value \(nipype.interfaces.base.Str attribute\), 132](#)
[default_value_type \(nipype.interfaces.base.InputMultiPath attribute\), 102](#)
[default_value_type \(nipype.interfaces.base.MultiPath attribute\), 115](#)
[default_value_type \(nipype.interfaces.base.OutputMultiPath attribute\), 119](#)
[default_value_type \(nipype.interfaces.base.Str attribute\), 132](#)
[DegreeCentrality, 608](#)
[Denoise, 842](#)
[DenoiseImage, 694](#)
[Despike, 610](#)
[Detrend, 611](#)
[Dewisp, 732](#)
[Dfs, 733](#)
[DICOMConvert, 902](#)
[DicomImport, 1596](#)
[DicomToNrrdConverter, 1475](#)
[dictcopy\(\) \(nipype.interfaces.base.Bunch method\), 81](#)
[DiffusionTensorScalarMeasurements, 1483](#)
[DiffusionTensorStreamlineTrack, 1238](#)
[DiffusionWeightedVolumeMasking, 1484](#)
[DilateImage, 1032, 1404](#)
[DilateMask, 1405](#)
[DipyBaseInterface, 842](#)
[DipyDiffusionInterface, 842](#)
[Directions2Amplitude, 1233](#)
[disconnect\(\) \(nipype.pipeline.engine.workflows.Workflow method\), 165](#)
[Distance, 565, 574](#)
[DistanceMap, 995](#)
[DistanceMaps, 1405](#)
[DT2NIFTI, 756](#)
[DTI, 848](#)
[DTIInputSpec, 1352](#)
[dtiestim, 1353](#)
[DTIExport, 1478](#)
[DTIFit, 770, 993](#)
[DTIImportSpec, 1479](#)
[dtiprocess, 1356](#)
[DTIRetop, 639](#)
[DTITracker, 830](#)
[DTLUTGen, 771](#)
[DTMetric, 772](#)
[Dump, 1154](#)
[DumpBinaryTrainingVectors, 1406](#)
[DWI2SphericalHarmonicsImage, 1232](#)
[DWI2Tensor, 1218](#)
[DWICompare, 1348](#)
[DWIConvert, 1350](#)
[DWIJointRicianLMMSEFilter, 1479](#)
[DWIRicianLMMSEFilter, 1481](#)
[DWISimpleCompare, 1349](#)
[DWIToDTIEstimation, 1482](#)
[DwiTool, 1278](#)
[DWIUnbiasedNonLocalMeansFilter, 1512](#)
[DynamicTraitedSpec \(class in nipype.interfaces.base\), 92](#)
- ## E
- [ECM, 612](#)

[Eddy](#), 1014
[EddyCorrect](#), 1017
[edit_traits\(\)](#) (nipype.interfaces.base.BaseInterfaceInputSpec method), 67
[edit_traits\(\)](#) (nipype.interfaces.base.BaseTraitedSpec method), 76
[edit_traits\(\)](#) (nipype.interfaces.base.CommandLineInputSpec method), 87
[edit_traits\(\)](#) (nipype.interfaces.base.DynamicTraitedSpec method), 96
[edit_traits\(\)](#) (nipype.interfaces.base.MpiCommandLineInputSpec method), 110
[edit_traits\(\)](#) (nipype.interfaces.base.StdOutCommandLineInputSpec method), 127
[edit_traits\(\)](#) (nipype.interfaces.base.TraitedSpec method), 139
[edit_traits\(\)](#) (nipype.interfaces.matlab.MatlabInputSpec method), 150
[editable_traits\(\)](#) (nipype.interfaces.base.BaseInterfaceInputSpec method), 68
[editable_traits\(\)](#) (nipype.interfaces.base.BaseTraitedSpec method), 77
[editable_traits\(\)](#) (nipype.interfaces.base.CommandLineInputSpec method), 88
[editable_traits\(\)](#) (nipype.interfaces.base.DynamicTraitedSpec method), 96
[editable_traits\(\)](#) (nipype.interfaces.base.MpiCommandLineInputSpec method), 110
[editable_traits\(\)](#) (nipype.interfaces.base.StdOutCommandLineInputSpec method), 128
[editable_traits\(\)](#) (nipype.interfaces.base.TraitedSpec method), 139
[editable_traits\(\)](#) (nipype.interfaces.matlab.MatlabInputSpec method), 150
[editor](#) (nipype.interfaces.base.InputMultiPath attribute), 102
[editor](#) (nipype.interfaces.base.MultiPath attribute), 115
[editor](#) (nipype.interfaces.base.OutputMultiPath attribute), 119
[editor](#) (nipype.interfaces.base.Str attribute), 132
[EditTransform](#), 857
[EditWMwithAseg](#), 903
[EM](#), 1307
[EMRegister](#), 932
[EMSegmentCommandLine](#), 1551
[EMSegmentTransformToNewFormat](#), 1562
[EngineBase](#) (class in nipype.pipeline.engine.base), 155
[EPIDeWarp](#), 1013
[EpiReg](#), 1018
[Erode](#), 1219
[ErodeImage](#), 1034, 1407
[error\(\)](#) (nipype.interfaces.base.InputMultiPath method), 102
[error\(\)](#) (nipype.interfaces.base.MultiPath method), 116
[error\(\)](#) (nipype.interfaces.base.OutputMultiPath method), 119
[error\(\)](#) (nipype.interfaces.base.Str method), 133
[ErrorMap](#), 565
[ESLR](#), 1450
[EstimateContrast](#), 1331, 1563
[EstimateFOD](#), 1258
[EstimateModel](#), 1564
[EstimateResponseForSH](#), 1235
[EstimateResponseSH](#), 845
[Eval](#), 647
[eval_spec_connect_function\(\)](#) (in module nipype.pipeline.engine.utils), 162
[expand_iterables\(\)](#) (in module nipype.pipeline.engine.utils), 162
[ExpertAutomatedRegistration](#), 1519
[export\(\)](#) (nipype.pipeline.engine.workflows.Workflow method), 165
[export_graph\(\)](#) (in module nipype.pipeline.engine.utils), 162
[Extract](#), 1155
[ExtractMainComponent](#), 948
[extractNrrdVectorIndex](#), 1361
[ExtractROI](#), 1109
[ExtractSkeleton](#), 1499

F

[FastInputSpec](#), 1565
[FAST](#), 1081
[fast_validate](#) (nipype.interfaces.base.Str attribute), 133
[fcsv_to_hdf5](#), 1471
[FEAT](#), 1053
[FEATModel](#), 1053
[FEATRegister](#), 1054
[FeatureExtractor](#), 1027
[fiberprocess](#), 1392
[fiberstats](#), 1391
[fibertrack](#), 1394
[FiducialRegistration](#), 1543
[filename\(\)](#) (nipype.sphinxext.plot_workflow.ImageFile method), 169
[filenames\(\)](#) (nipype.sphinxext.plot_workflow.ImageFile method), 169
[fileno\(\)](#) (nipype.interfaces.base.Stream method), 134
[FillLesions](#), 1312
[FILMGLS](#), 1054
[FilterRegressor](#), 1110
[FilterTracks](#), 1241
[Fim](#), 613
[FindCenterOfBrain](#), 1466
[FindShPeaks](#), 1236
[FindTheBiggest](#), 996
[FIRST](#), 1083

- FitAsl, 1275
- FitDwi, 1281
- FitGLM, 1332
- FitMSParams, 904
- FitQt1, 1285
- FitTensor, 1261
- FixTopology, 949
- FLAMEO, 1057
- FlippedDifference, 1408
- FLIRT, 1085
- FmriRealign4d, 1333
- FNIRT, 1088
- format_dot() (in module nipy.pipeline.engine.utils), 162
- format_node() (in module nipy.pipeline.engine.utils), 162
- Fourier, 614
- FramewiseDisplacement, 554
- FreeSurferSource, 1629
- FUGUE, 1092
- full_info() (nipy.interfaces.base.InputMultiPath method), 102
- full_info() (nipy.interfaces.base.MultiPath method), 116
- full_info() (nipy.interfaces.base.OutputMultiPath method), 119
- full_info() (nipy.interfaces.base.Str method), 133
- fullname (nipy.pipeline.engine.base.EngineBase attribute), 155
- fullname (nipy.pipeline.engine.nodes.JoinNode attribute), 157
- fullname (nipy.pipeline.engine.nodes.MapNode attribute), 159
- fullname (nipy.pipeline.engine.nodes.Node attribute), 161
- fullname (nipy.pipeline.engine.workflows.Workflow attribute), 165
- Function, 1606
- FuseSegmentations, 859
- FuzzyOverlap, 566, 574
- FWHMx, 648
- G**
- GaussianBlurImageFilter, 1496
- Generate5tt, 1269
- generate_expanded_graph() (in module nipy.pipeline.engine.utils), 162
- GenerateBrainClippedImage, 1409
- GenerateCsfClippedFromClassifiedImage, 1399
- GenerateDirections, 1237
- GenerateEdgeMapImage, 1345
- GenerateLabelMapFromProbabilityMap, 1467
- GeneratePurePlugMask, 1346
- GenerateSummedGradientImage, 1409
- GenerateTestImage, 1410
- GenerateWhiteMatterMask, 1220
- Gennlxfm, 1158
- GenWarpFields, 665
- get() (nipy.interfaces.base.BaseInterfaceInputSpec method), 68
- get() (nipy.interfaces.base.BaseTraitedSpec method), 77
- get() (nipy.interfaces.base.Bunch method), 81
- get() (nipy.interfaces.base.CommandLineInputSpec method), 88
- get() (nipy.interfaces.base.DynamicTraitedSpec method), 96
- get() (nipy.interfaces.base.MpiCommandLineInputSpec method), 110
- get() (nipy.interfaces.base.OutputMultiPath method), 119
- get() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 128
- get() (nipy.interfaces.base.TraitedSpec method), 139
- get() (nipy.interfaces.matlab.MatlabInputSpec method), 150
- get_all_files() (in module nipy.pipeline.engine.utils), 162
- get_default_value() (nipy.interfaces.base.InputMultiPath method), 102
- get_default_value() (nipy.interfaces.base.MultiPath method), 116
- get_default_value() (nipy.interfaces.base.OutputMultiPath method), 119
- get_default_value() (nipy.interfaces.base.Str method), 133
- get_dependencies() (in module nipy.interfaces.base), 143
- get_editor() (nipy.interfaces.base.InputMultiPath method), 103
- get_editor() (nipy.interfaces.base.MultiPath method), 116
- get_editor() (nipy.interfaces.base.OutputMultiPath method), 120
- get_editor() (nipy.interfaces.base.Str method), 133
- get_hashval() (nipy.interfaces.base.BaseInterfaceInputSpec method), 68
- get_hashval() (nipy.interfaces.base.BaseTraitedSpec method), 77
- get_hashval() (nipy.interfaces.base.CommandLineInputSpec method), 88
- get_hashval() (nipy.interfaces.base.DynamicTraitedSpec method), 96
- get_hashval() (nipy.interfaces.base.MpiCommandLineInputSpec method), 110
- get_hashval() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 128
- get_hashval() (nipy.interfaces.base.TraitedSpec method), 139

- method), 139
 - get_hashval() (nipy.interfaces.matlab.MatlabInputSpec method), 150
 - get_levels() (in module nipy.pipeline.engine.utils), 162
 - get_matlab_command() (in module nipy.interfaces.matlab), 154
 - get_max_resources_used() (in module nipy.interfaces.base), 143
 - get_node() (nipy.pipeline.engine.workflows.Workflow method), 165
 - get_output() (nipy.pipeline.engine.nodes.JoinNode method), 157
 - get_output() (nipy.pipeline.engine.nodes.MapNode method), 159
 - get_output() (nipy.pipeline.engine.nodes.Node method), 161
 - get_print_name() (in module nipy.pipeline.engine.utils), 162
 - get_subnodes() (nipy.pipeline.engine.nodes.MapNode method), 159
 - get_traitsfree() (nipy.interfaces.base.BaseInterfaceInputSpec method), 68
 - get_traitsfree() (nipy.interfaces.base.BaseTraitedSpec method), 77
 - get_traitsfree() (nipy.interfaces.base.CommandLineInputSpec method), 88
 - get_traitsfree() (nipy.interfaces.base.DynamicTraitedSpec method), 97
 - get_traitsfree() (nipy.interfaces.base.MpiCommandLineInputSpec method), 111
 - get_traitsfree() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 128
 - get_traitsfree() (nipy.interfaces.base.TraitedSpec method), 140
 - get_traitsfree() (nipy.interfaces.matlab.MatlabInputSpec method), 151
 - get_value() (nipy.interfaces.base.InputMultiPath method), 103
 - get_value() (nipy.interfaces.base.MultiPath method), 116
 - get_value() (nipy.interfaces.base.OutputMultiPath method), 120
 - get_value() (nipy.interfaces.base.Str method), 133
 - get_wf_formats() (in module nipy.sphinxext.plot_workflow), 169
 - GLM, 1059
 - GLMFit, 866
 - GradientAnisotropicDiffusion, 1497
 - GradientAnisotropicDiffusionImageFilter, 1411
 - GraphError (class in nipy.sphinxext.plot_workflow), 169
 - GrayscaleFillHoleImageFilter, 1502
 - GrayscaleGrindPeakImageFilter, 1503
 - GrayscaleModelMaker, 1554
 - GroupAndStack, 1618
 - gtractAnisotropyMap, 1362
 - gtractAverageBvalues, 1363
 - gtractClipAnisotropy, 1364
 - gtractConcatDwi, 1367
 - gtractCopyImageOrientation, 1368
 - gtractCoRegAnatomy, 1365
 - gtractCoregBvalues, 1369
 - gtractCostFastMarching, 1371
 - gtractCreateGuideFiber, 1373
 - gtractFastMarchingTracking, 1374
 - gtractFiberTracking, 1375
 - gtractImageConformity, 1378
 - gtractInvertBSplineTransform, 1379
 - gtractInvertDisplacementField, 1380
 - gtractInvertRigidTransform, 1381
 - gtractResampleAnisotropy, 1382
 - gtractResampleB0, 1383
 - gtractResampleCodeImage, 1384
 - gtractResampleDWIInPlace, 1385
 - gtractResampleFibers, 1387
 - gtractTensor, 1388
 - gtractTransformToDisplacementField, 1389
 - Gunzip, 575
- ## H
- HammerAttributeCreator, 1412
 - HARDIMat, 832
 - has_items (nipy.interfaces.base.InputMultiPath attribute), 103
 - has_items (nipy.interfaces.base.MultiPath attribute), 116
 - has_items (nipy.interfaces.base.OutputMultiPath attribute), 120
 - has_items (nipy.interfaces.base.Str attribute), 133
 - has_metadata() (nipy.interfaces.base.BaseInterfaceInputSpec method), 68
 - has_metadata() (nipy.interfaces.base.BaseTraitedSpec method), 77
 - has_metadata() (nipy.interfaces.base.CommandLineInputSpec method), 88
 - has_metadata() (nipy.interfaces.base.DynamicTraitedSpec method), 97
 - has_metadata() (nipy.interfaces.base.MpiCommandLineInputSpec method), 111
 - has_metadata() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 128
 - has_metadata() (nipy.interfaces.base.TraitedSpec method), 140
 - has_metadata() (nipy.interfaces.matlab.MatlabInputSpec method), 151
 - has_traits_interface() (nipy.interfaces.base.BaseInterfaceInputSpec method), 68

[has_traits_interface\(\) \(nipype.interfaces.base.BaseTraitedSpec method\), 77](#)
[has_traits_interface\(\) \(nipype.interfaces.base.CommandLineInputSpec method\), 88](#)
[has_traits_interface\(\) \(nipype.interfaces.base.DynamicTraitedSpec method\), 97](#)
[has_traits_interface\(\) \(nipype.interfaces.base.MpiCommandLineInputSpec method\), 111](#)
[has_traits_interface\(\) \(nipype.interfaces.base.StdOutCommandLineInputSpec method\), 128](#)
[has_traits_interface\(\) \(nipype.interfaces.base.TraitedSpec method\), 140](#)
[has_traits_interface\(\) \(nipype.interfaces.matlab.MatlabInputSpec method\), 151](#)
[hash_exists\(\) \(nipype.pipeline.engine.nodes.JoinNode method\), 157](#)
[hash_exists\(\) \(nipype.pipeline.engine.nodes.MapNode method\), 159](#)
[hash_exists\(\) \(nipype.pipeline.engine.nodes.Node method\), 161](#)
[help\(\) \(nipype.interfaces.base.BaseInterface class method\), 63](#)
[help\(\) \(nipype.interfaces.base.CommandLine class method\), 83](#)
[help\(\) \(nipype.interfaces.base.Interface class method\), 104](#)
[help\(\) \(nipype.interfaces.base.MpiCommandLine method\), 105](#)
[help\(\) \(nipype.interfaces.base.SEMLikeCommandLine method\), 121](#)
[help\(\) \(nipype.interfaces.base.StdOutCommandLine method\), 122](#)
[help\(\) \(nipype.interfaces.matlab.MatlabCommand method\), 145](#)
[help\(\) \(nipype.pipeline.engine.nodes.JoinNode method\), 157](#)
[help\(\) \(nipype.pipeline.engine.nodes.MapNode method\), 159](#)
[help\(\) \(nipype.pipeline.engine.nodes.Node method\), 161](#)
[Hemisplit, 735](#)
[Hist, 616](#)
[HistogramMatching, 1500](#)
[HistogramMatchingFilter, 1347](#)
I
[ICA_AROMA, 989](#)
[ICC, 559](#)
[IdentityInterface, 1601](#)
[Image2Voxel, 756](#)
[ImageFile \(class in nipype.sphinxext.plot_workflow\), 169](#)
[ImageLabelCombine, 1501](#)
[ImageMaths, 1111](#)
[ImageMeants, 1112](#)
[ImageRegionPlotter, 1468](#)
[ImageStats, 808, 1113](#)
[info\(\) \(nipype.interfaces.base.InputMultiPath method\), 103](#)
[info\(\) \(nipype.interfaces.base.MultiPath method\), 116](#)
[info_spec \(nipype.interfaces.base.OutputMultiPath method\), 120](#)
[info_text \(nipype.interfaces.base.Str method\), 133](#)
[info_text \(nipype.interfaces.base.InputMultiPath attribute\), 103](#)
[info_text \(nipype.interfaces.base.MultiPath attribute\), 117](#)
[info_text \(nipype.interfaces.base.OutputMultiPath attribute\), 120](#)
[info_text \(nipype.interfaces.base.Str attribute\), 134](#)
[info_trait \(nipype.interfaces.base.InputMultiPath attribute\), 103](#)
[info_trait \(nipype.interfaces.base.MultiPath attribute\), 117](#)
[info_trait \(nipype.interfaces.base.OutputMultiPath attribute\), 120](#)
[init\(\) \(nipype.interfaces.base.InputMultiPath method\), 103](#)
[init\(\) \(nipype.interfaces.base.MultiPath method\), 117](#)
[init\(\) \(nipype.interfaces.base.OutputMultiPath method\), 120](#)
[init\(\) \(nipype.interfaces.base.Str method\), 134](#)
[inner_traits\(\) \(nipype.interfaces.base.InputMultiPath method\), 103](#)
[inner_traits\(\) \(nipype.interfaces.base.MultiPath method\), 117](#)
[inner_traits\(\) \(nipype.interfaces.base.OutputMultiPath method\), 120](#)
[inner_traits\(\) \(nipype.interfaces.base.Str method\), 134](#)
[input_spec \(nipype.interfaces.base.BaseInterface attribute\), 63](#)
[input_spec \(nipype.interfaces.base.CommandLine attribute\), 83](#)
[input_spec \(nipype.interfaces.base.Interface attribute\), 104](#)
[input_spec \(nipype.interfaces.base.MpiCommandLine attribute\), 105](#)
[input_spec \(nipype.interfaces.base.SEMLikeCommandLine attribute\), 121](#)
[input_spec \(nipype.interfaces.base.StdOutCommandLine attribute\), 122](#)
[input_spec \(nipype.interfaces.matlab.MatlabCommand attribute\), 145](#)
[InputMultiPath \(class in nipype.interfaces.base\), 100](#)
[inputs \(nipype.pipeline.engine.base.EngineBase attribute\), 155](#)
[inputs \(nipype.pipeline.engine.nodes.JoinNode attribute\), 157](#)
[inputs \(nipype.pipeline.engine.nodes.MapNode attribute\), 159](#)

- inputs (nipyre.pipeline.engine.nodes.Node attribute), 161
 - inputs (nipyre.pipeline.engine.workflows.Workflow attribute), 165
 - insertMidACPCpoint, 1472
 - IntensityDifferenceMetric, 1528
 - Interface (class in nipyre.interfaces.base), 103
 - interface (nipyre.pipeline.engine.nodes.JoinNode attribute), 157
 - interface (nipyre.pipeline.engine.nodes.MapNode attribute), 159
 - interface (nipyre.pipeline.engine.nodes.Node attribute), 161
 - InterfaceResult (class in nipyre.interfaces.base), 104
 - InvWarp, 1114
 - IOBase, 1631
 - is_mapped (nipyre.interfaces.base.InputMultiPath attribute), 103
 - is_mapped (nipyre.interfaces.base.MultiPath attribute), 117
 - is_mapped (nipyre.interfaces.base.OutputMultiPath attribute), 120
 - is_mapped (nipyre.interfaces.base.Str attribute), 134
 - is_valid() (nipyre.interfaces.base.InputMultiPath method), 103
 - is_valid() (nipyre.interfaces.base.MultiPath method), 117
 - is_valid() (nipyre.interfaces.base.OutputMultiPath method), 120
 - is_valid() (nipyre.interfaces.base.Str method), 134
 - IsotropicSmooth, 1035
 - items() (nipyre.interfaces.base.BaseInterfaceInputSpec method), 68
 - items() (nipyre.interfaces.base.BaseTraitedSpec method), 77
 - items() (nipyre.interfaces.base.Bunch method), 81
 - items() (nipyre.interfaces.base.CommandLineInputSpec method), 88
 - items() (nipyre.interfaces.base.DynamicTraitedSpec method), 97
 - items() (nipyre.interfaces.base.MpiCommandLineInputSpec method), 111
 - items() (nipyre.interfaces.base.StdOutCommandLineInputSpec method), 128
 - items() (nipyre.interfaces.base.TraitedSpec method), 140
 - items() (nipyre.interfaces.matlab.MatlabInputSpec method), 151
 - items_event() (nipyre.interfaces.base.InputMultiPath method), 103
 - items_event() (nipyre.interfaces.base.MultiPath method), 117
 - items_event() (nipyre.interfaces.base.OutputMultiPath method), 120
 - iteritems() (nipyre.interfaces.base.Bunch method), 81
 - itername (nipyre.pipeline.engine.base.EngineBase attribute), 155
 - itername (nipyre.pipeline.engine.nodes.JoinNode attribute), 157
 - itername (nipyre.pipeline.engine.nodes.MapNode attribute), 159
 - itername (nipyre.pipeline.engine.nodes.Node attribute), 161
 - itername (nipyre.pipeline.engine.workflows.Workflow attribute), 165
- ## J
- Jacobian, 951
 - JistBrainMgdmSegmentation, 1187
 - JistBrainMp2rageDuraEstimation, 1189
 - JistBrainMp2rageSkullStripping, 1190
 - JistBrainPartialVolumeFilter, 1191
 - JistCortexSurfaceMeshInflation, 1192
 - JistIntensityMp2rageMasking, 1194
 - JistLaminarProfileCalculator, 1195
 - JistLaminarProfileGeometry, 1196
 - JistLaminarProfileSampling, 1197
 - JistLaminarROI Averaging, 1198
 - JistLaminarVolumetricLayering, 1199
 - JoinNode (class in nipyre.pipeline.engine.nodes), 156
 - joinsource (nipyre.pipeline.engine.nodes.JoinNode attribute), 157
 - JointFusion, 695
 - JointHistogram, 1469
 - JSONFileGrabber, 1631
 - JSONFileSink, 1632
- ## K
- KellyKapowski, 697
- ## L
- L2Model, 1061
 - Label2Annot, 870
 - Label2Label, 871
 - Label2Vol, 873
 - LabelConfig, 1255
 - LabelFusion, 1310
 - LabelMapSmoothing, 1555
 - landmarksConstellationAligner, 1472
 - landmarksConstellationWeights, 1473
 - LaplacianThickness, 699
 - Level1Design, 1062, 1566
 - LFCD, 617
 - LinearRegistration, 1521
 - LinRecon, 801
 - list_node_names() (nipyre.pipeline.engine.workflows.Workflow method), 165
 - load() (nipyre.pipeline.engine.base.EngineBase method), 155
 - load() (nipyre.pipeline.engine.nodes.JoinNode method), 157

- load() (nipyre.pipeline.engine.nodes.MapNode method), 159
- load() (nipyre.pipeline.engine.nodes.Node method), 161
- load() (nipyre.pipeline.engine.workflows.Workflow method), 166
- load_inputs_from_json() (nipyre.interfaces.base.BaseInterface method), 63
- load_inputs_from_json() (nipyre.interfaces.base.CommandLine method), 83
- load_inputs_from_json() (nipyre.interfaces.base.MpiCommandLine method), 105
- load_inputs_from_json() (nipyre.interfaces.base.SEMLikeCommandLine method), 121
- load_inputs_from_json() (nipyre.interfaces.base.StdOutCommandLine method), 122
- load_inputs_from_json() (nipyre.interfaces.matlab.MatlabCommand method), 145
- load_template() (in module nipyre.interfaces.base), 143
- LookupMeta, 1618
- ## M
- make_output_dir() (in module nipyre.pipeline.engine.utils), 162
- MakeAverageSubject, 962
- MakeDyadicVectors, 997
- MakeSurfaces, 963
- MapNode (class in nipyre.pipeline.engine.nodes), 158
- mark_wf_labels() (in module nipyre.sphinxext.plot_workflow), 169
- Maskave, 618
- MaskScalarVolume, 1491
- MaskTool, 651
- Math, 1159
- MathsCommand, 1036, 1317
- Matlab2CSV, 575
- MatlabCommand (class in nipyre.interfaces.matlab), 144
- MatlabInputSpec (class in nipyre.interfaces.matlab), 146
- maxcurvature, 1390
- MaxImage, 1037
- MaxnImage, 1038
- MCFLIRT, 1095
- MeanImage, 1039
- Means, 619
- MedianFilter3D, 1225
- MedianImage, 1040
- MedianImageFilter, 1498
- MedicAlgorithmImageCalculator, 1201
- MedicAlgorithmLesionToads, 1202
- MedicAlgorithmMipavReorient, 1205
- MedicAlgorithmN3, 1206
- MedicAlgorithmSPECTRE2010, 1208
- MedicAlgorithmThresholdToBinaryMask, 1212
- MELODIC, 1063
- Memory (class in nipyre.caching), 60
- Merge, 653, 1116, 1318, 1602
- merge_bundles() (in module nipyre.pipeline.engine.utils), 162
- merge_dict() (in module nipyre.pipeline.engine.utils), 162
- MergeCNetworks, 820
- MergeCSVFiles, 576
- MergeModels, 1556
- MergeNifti, 1619
- MergeROIs, 577
- MergeSDS, 401
- Mesh2PVE, 1270
- MeshFix, 641
- MeshWarp, 562
- metadata (nipyre.interfaces.base.InputMultiPath attribute), 103
- metadata (nipyre.interfaces.base.MultiPath attribute), 117
- metadata (nipyre.interfaces.base.OutputMultiPath attribute), 120
- metadata (nipyre.interfaces.base.Str attribute), 134
- MinImage, 1041
- MNIBiasCorrection, 905
- ModelFit, 774
- ModelMaker, 1557
- ModelToLabelMap, 1560
- modify_paths() (in module nipyre.pipeline.engine.utils), 163
- ModifyAffine, 577
- MotionOutliers, 1117
- MpiCommandLine (class in nipyre.interfaces.base), 104
- MpiCommandLineInputSpec (class in nipyre.interfaces.base), 106
- MPRtoMNI305, 933
- MRConvert, 1221
- MRICConvert, 907
- MRIFill, 951
- MRIMarchingCubes, 952
- MRIPretess, 953
- MRIsCALabel, 912
- MRIsCalc, 956
- MRIsCombine, 957
- MRIsConvert, 958
- MRIsExpand, 960
- MRIsInflate, 961
- MRISPreproc, 875
- MRISPreprocReconAll, 877
- MRITessellate, 955
- MRMultiply, 1222
- MRTransform, 1223
- MRTrix2TrackVis, 1217
- MRTrix3Base, 1253
- MRTrixViewer, 1225

MS_LDA, 879
 MultiImageMaths, 1041
 MultiPath (class in nipype.interfaces.base), 114
 MultipleRegressDesign, 1067
 MultipleRegressionDesign, 1568
 MultiplyImages, 711
 MultiplyScalarVolumes, 1492
 MultiResolutionAffineRegistration, 1523
 MySQLSink, 1632

N

N4BiasFieldCorrection, 701
 N4ITKBiasFieldCorrection, 1504
 NeighborhoodMean, 1413
 NeighborhoodMedian, 1414
 NetworkBasedStatistic, 821
 NetworkXMetrics, 823
 NewSegment, 1580
 NIFTIDT2Camino, 757
 NiftiGeneratorBase, 1620
 NiftyFitCommand, 1278
 NiftyRegCommand, 1289
 NiftySegCommand, 1307
 nipype.algorithms.confounds (module), 557
 nipype.algorithms.icc (module), 559
 nipype.algorithms.misc (module), 582
 nipype.algorithms.modelgen (module), 590
 nipype.caching.memory (module), 60
 nipype.interfaces.base (module), 62
 nipype.interfaces.brainsuite.brainsuite (module), 746
 nipype.interfaces.cmtk.cmtk (module), 818
 nipype.interfaces.cmtk.nbs (module), 822
 nipype.interfaces.cmtk.nx (module), 824
 nipype.interfaces.cmtk.parcellation (module), 826
 nipype.interfaces.dcmstack (module), 1620
 nipype.interfaces.dipy.preprocess (module), 844
 nipype.interfaces.freesurfer.utils (module), 988
 nipype.interfaces.io (module), 1639
 nipype.interfaces.matlab (module), 144
 nipype.interfaces.minc.base (module), 1137
 nipype.interfaces.minc.testdata (module), 1186
 nipype.interfaces.mrtrix.convert (module), 1217
 nipype.interfaces.mrtrix.tensors (module), 1238
 nipype.interfaces.niftyreg.base (module), 1289
 nipype.interfaces.vtkbase (module), 1651
 nipype.pipeline.engine.base (module), 155
 nipype.pipeline.engine.nodes (module), 156
 nipype.pipeline.engine.utils (module), 161
 nipype.pipeline.engine.workflows (module), 164
 nipype.sphinxext.plot_workflow (module), 167
 nipype.workflows.data (module), 203
 nipype.workflows.dmri.camino.connectivity_mapping (module), 205
 nipype.workflows.dmri.camino.diffusion (module), 206

nipype.workflows.dmri.camino.group_connectivity (module), 207
 nipype.workflows.dmri.connectivity.group_connectivity (module), 208
 nipype.workflows.dmri.connectivity.nx (module), 211
 nipype.workflows.dmri.dipy.denoise (module), 213
 nipype.workflows.dmri.fsl.artifacts (module), 214
 nipype.workflows.dmri.fsl.dti (module), 231
 nipype.workflows.dmri.fsl.epi (module), 235
 nipype.workflows.dmri.fsl.tbss (module), 247
 nipype.workflows.dmri.fsl.utils (module), 259
 nipype.workflows.dmri.mrtrix.connectivity_mapping (module), 265
 nipype.workflows.dmri.mrtrix.diffusion (module), 266
 nipype.workflows.dmri.mrtrix.group_connectivity (module), 268
 nipype.workflows.fMRI.fsl.estimate (module), 269
 nipype.workflows.fMRI.fsl.preprocess (module), 273
 nipype.workflows.fMRI.spm.preprocess (module), 282
 nipype.workflows.misc.utils (module), 289
 nipype.workflows.rsfmri.fsl.resting (module), 291
 nipype.workflows.smri.ants.ANTSBuildTemplate (module), 297
 nipype.workflows.smri.ants.antsRegistrationBuildTemplate (module), 299
 nipype.workflows.smri.freesurfer.autorecon1 (module), 300
 nipype.workflows.smri.freesurfer.autorecon2 (module), 300
 nipype.workflows.smri.freesurfer.bem (module), 300
 nipype.workflows.smri.freesurfer.recon (module), 302
 nipype.workflows.smri.freesurfer.utils (module), 304
 nipype.workflows.smri.niftyreg.groupwise (module), 310
 NipypeInterfaceError (class in nipype.interfaces.base), 117
 NlpFit, 1164
 Node (class in nipype.pipeline.engine.nodes), 159
 NonSteadyStateDetector, 555
 Norm, 1165
 Normalize, 914, 1582
 Normalize12, 1584
 NormalizeProbabilityMapSet, 578
 Notes, 654
 num_subnodes() (nipype.pipeline.engine.nodes.MapNode method), 159

O

ODFRecon, 833
 ODFTracker, 835
 on_trait_change() (nipype.interfaces.base.BaseInterfaceInputSpec method), 68
 on_trait_change() (nipype.interfaces.base.BaseTrainedSpec method), 77

- [on_trait_change\(\) \(nipype.interfaces.base.CommandLineInputSpec method\)](#), 88
[on_trait_change\(\) \(nipype.interfaces.base.DynamicTraitedSpec method\)](#), 97
[on_trait_change\(\) \(nipype.interfaces.base.MpiCommandLineInputSpec method\)](#), 111
[on_trait_change\(\) \(nipype.interfaces.base.StdOutCommandLineInputSpec method\)](#), 128
[on_trait_change\(\) \(nipype.interfaces.base.TraitedSpec method\)](#), 140
[on_trait_change\(\) \(nipype.interfaces.matlab.MatlabInputSpec method\)](#), 151
[on_trait_event\(\) \(nipype.interfaces.base.BaseInterfaceInputSpec method\)](#), 69
[on_trait_event\(\) \(nipype.interfaces.base.BaseTraitedSpec method\)](#), 78
[on_trait_event\(\) \(nipype.interfaces.base.CommandLineInputSpec method\)](#), 89
[on_trait_event\(\) \(nipype.interfaces.base.DynamicTraitedSpec method\)](#), 97
[on_trait_event\(\) \(nipype.interfaces.base.MpiCommandLineInputSpec method\)](#), 111
[on_trait_event\(\) \(nipype.interfaces.base.StdOutCommandLineInputSpec method\)](#), 128
[on_trait_event\(\) \(nipype.interfaces.base.TraitedSpec method\)](#), 140
[on_trait_event\(\) \(nipype.interfaces.matlab.MatlabInputSpec method\)](#), 151
[OneSampleTTest](#), 880
[OneSampleTTestDesign](#), 1569
[OrientScalarVolume](#), 1477
[OtsuThresholdImageFilter](#), 1513
[OtsuThresholdSegmentation](#), 1527
[out_of_date\(\) \(in module nipype.sphinxext.plot_workflow\)](#), 169
[OutlierCount](#), 620
[output_dir\(\) \(nipype.pipeline.engine.nodes.JoinNode method\)](#), 157
[output_dir\(\) \(nipype.pipeline.engine.nodes.MapNode method\)](#), 159
[output_dir\(\) \(nipype.pipeline.engine.nodes.Node method\)](#), 161
[output_spec \(nipype.interfaces.base.BaseInterface attribute\)](#), 63
[output_spec \(nipype.interfaces.base.CommandLine attribute\)](#), 83
[output_spec \(nipype.interfaces.base.Interface attribute\)](#), 104
[output_spec \(nipype.interfaces.base.MpiCommandLine attribute\)](#), 105
[output_spec \(nipype.interfaces.base.SEMLikeCommandLine attribute\)](#), 121
[output_spec \(nipype.interfaces.base.StdOutCommandLine attribute\)](#), 122
[output_spec \(nipype.interfaces.matlab.MatlabCommand attribute\)](#), 145
[OutputMultiPath \(class in nipype.interfaces.base\)](#), 117
[outputs \(nipype.pipeline.engine.base.EngineBase attribute\)](#), 155
[outputs \(nipype.pipeline.engine.nodes.JoinNode attribute\)](#), 157
[outputs \(nipype.pipeline.engine.nodes.MapNode attribute\)](#), 159
[outputs \(nipype.pipeline.engine.nodes.Node attribute\)](#), 161
[outputs \(nipype.pipeline.engine.workflows.Workflow attribute\)](#), 166
[Overlap](#), 567, 578
[Overlay](#), 1118

P

[P2PDistance](#), 562
[Paint](#), 934
[PairedTTestDesign](#), 1570
[partname](#), 825
[ParcellationStats](#), 965
[parsePRNDir](#), 915
[PatchMatch](#), 1323
[PercentileImage](#), 1043
[PETPVC](#), 1647
[PETStandardUptakeValueComputation](#), 1529
[Pialmesh](#), 736
[PickAtlas](#), 579
[PicoPDFs](#), 777
[Pik](#), 1167
[PipeFunc \(class in nipype.caching.memory\)](#), 8, 61
[PlotMotionParams](#), 1120
[PlotTimeSeries](#), 1121
[PointsWarp](#), 855
[PowerSpectrum](#), 1123
[PRELUDE](#), 1097
[PrepareFieldmap](#), 1020
[print_traits\(\) \(nipype.interfaces.base.BaseInterfaceInputSpec method\)](#), 69
[print_traits\(\) \(nipype.interfaces.base.BaseTraitedSpec method\)](#), 78
[print_traits\(\) \(nipype.interfaces.base.CommandLineInputSpec method\)](#), 89
[print_traits\(\) \(nipype.interfaces.base.DynamicTraitedSpec method\)](#), 98
[print_traits\(\) \(nipype.interfaces.base.MpiCommandLineInputSpec method\)](#), 112
[print_traits\(\) \(nipype.interfaces.base.StdOutCommandLineInputSpec method\)](#), 129
[print_traits\(\) \(nipype.interfaces.base.TraitedSpec method\)](#), 141
[print_traits\(\) \(nipype.interfaces.matlab.MatlabInputSpec method\)](#), 152

ProbabilisticSphericallyDeconvolutedStreamlineTrack,
1242
ProbeVolumeWithModel, 1561
ProbTrackX, 998
ProbTrackX2, 1001
ProcStreamlines, 758
ProjThresh, 1005
Pvc, 738

Q

QBallMX, 804
QualityIndex, 622
QwarpPlusMinus, 623

R

raise_exception() (nipy.interfaces.base.CommandLine
method), 83
raise_exception() (nipy.interfaces.base.MpiCommandLine
method), 105
raise_exception() (nipy.interfaces.base.SEMLikeCommandLine
method), 121
raise_exception() (nipy.interfaces.base.StdOutCommandLine
method), 122
raise_exception() (nipy.interfaces.matlab.MatlabCommand
method), 145
Randomise, 1068
RandomVol, 1213
read() (nipy.interfaces.base.Stream method), 134
read_log() (in module nipy.caching.memory), 62
Realign, 1585
ReconAll, 916
references_ (nipy.interfaces.base.BaseInterface at-
tribute), 63
references_ (nipy.interfaces.base.CommandLine
attribute), 83
references_ (nipy.interfaces.base.MpiCommandLine
attribute), 105
references_ (nipy.interfaces.base.SEMLikeCommandLine
attribute), 121
references_ (nipy.interfaces.base.StdOutCommandLine
attribute), 122
references_ (nipy.interfaces.matlab.MatlabCommand
attribute), 145
Refit, 655
RegAladin, 1290
RegAverage, 1296
RegF3D, 1292
Register, 935
RegisterAVItoTalairach, 936
Registration, 670, 856
RegJacobian, 1297
RegMeasure, 1298
RegResample, 1299
RegTools, 1301

RegTransform, 1302
RelabelHypointensities, 967
remove_coding() (in module
nipy.sphinxext.plot_workflow), 169
remove_nodes() (nipy.pipeline.engine.workflows.Workflow
method), 166
remove_trait() (nipy.interfaces.base.BaseInterfaceInputSpec
method), 69
remove_trait() (nipy.interfaces.base.BaseTraitedSpec
method), 78
remove_trait() (nipy.interfaces.base.CommandLineInputSpec
method), 89
remove_trait() (nipy.interfaces.base.DynamicTraitedSpec
method), 98
remove_trait() (nipy.interfaces.base.MpiCommandLineInputSpec
method), 112
remove_trait() (nipy.interfaces.base.StdOutCommandLineInputSpec
method), 129
remove_trait() (nipy.interfaces.base.TraitedSpec
method), 141
remove_trait() (nipy.interfaces.matlab.MatlabInputSpec
method), 152
remove_trait_listener() (nipy.interfaces.base.BaseInterfaceInputSpec
method), 69
remove_trait_listener() (nipy.interfaces.base.BaseTraitedSpec
method), 78
remove_trait_listener() (nipy.interfaces.base.CommandLineInputSpec
method), 89
remove_trait_listener() (nipy.interfaces.base.DynamicTraitedSpec
method), 98
remove_trait_listener() (nipy.interfaces.base.MpiCommandLineInputSpec
method), 112
remove_trait_listener() (nipy.interfaces.base.StdOutCommandLineInputSpec
method), 129
remove_trait_listener() (nipy.interfaces.base.TraitedSpec
method), 141
remove_trait_listener() (nipy.interfaces.matlab.MatlabInputSpec
method), 152
RemoveIntersection, 968
RemoveNeck, 969
Rename, 1603
render_figures() (in module
nipy.sphinxext.plot_workflow), 169
Reorient2Std, 1124
repr() (nipy.interfaces.base.InputMultiPath method),
103
repr() (nipy.interfaces.base.MultiPath method), 117
repr() (nipy.interfaces.base.OutputMultiPath method),
120
repr() (nipy.interfaces.base.Str method), 134
Resample, 656, 843, 921, 1169
ResampleDTIVolume, 1485
ResampleScalarVectorDWIVolume, 1506
ResampleScalarVolume, 1514

- reset_traits() (nipyype.interfaces.base.BaseInterfaceInputSpec.run_code() (in module nipyype.sphinxext.plot_workflow), method), 69
- reset_traits() (nipyype.interfaces.base.BaseTraitedSpec run_command() (in module nipyype.interfaces.base), 144 method), 78
- reset_traits() (nipyype.interfaces.base.CommandLineInputSpec method), 89
- reset_traits() (nipyype.interfaces.base.DynamicTraitedSpec method), 98
- reset_traits() (nipyype.interfaces.base.MpiCommandLineInputSpec method), 112
- reset_traits() (nipyype.interfaces.base.StdOutCommandLineInputSpec method), 129
- reset_traits() (nipyype.interfaces.base.TraitedSpec method), 141
- reset_traits() (nipyype.interfaces.matlab.MatlabInputSpec method), 152
- Reshape, 1174
- Reslice, 1597
- ResliceToReference, 1598
- ResponseSD, 1256
- RESTORE, 846
- result (nipyype.pipeline.engine.nodes.JoinNode attribute), 157
- result (nipyype.pipeline.engine.nodes.MapNode attribute), 159
- result (nipyype.pipeline.engine.nodes.Node attribute), 161
- Retroicor, 626
- RigidRegistration, 1524
- rm_all_but() (in module nipyype.caching.memory), 62
- RobustFOV, 1124
- RobustRegister, 922
- RobustStatisticsSegmenter, 1553
- RobustTemplate, 860
- ROIGen, 817
- ROIStats, 625
- run() (in module nipyype.sphinxext.plot_workflow), 169
- run() (nipyype.interfaces.base.BaseInterface method), 63
- run() (nipyype.interfaces.base.CommandLine method), 83
- run() (nipyype.interfaces.base.Interface method), 104
- run() (nipyype.interfaces.base.MpiCommandLine method), 105
- run() (nipyype.interfaces.base.SEMLikeCommandLine method), 121
- run() (nipyype.interfaces.base.StdOutCommandLine method), 122
- run() (nipyype.interfaces.matlab.MatlabCommand method), 145
- run() (nipyype.pipeline.engine.nodes.JoinNode method), 157
- run() (nipyype.pipeline.engine.nodes.MapNode method), 159
- run() (nipyype.pipeline.engine.nodes.Node method), 161
- run() (nipyype.pipeline.engine.workflows.Workflow method), 166
- S3DataGrabber, 1633
- SampleToSurface, 970
- save() (nipyype.pipeline.engine.base.EngineBase method), 155
- save() (nipyype.pipeline.engine.nodes.JoinNode method), 157
- save() (nipyype.pipeline.engine.nodes.MapNode method), 159
- save() (nipyype.pipeline.engine.nodes.Node method), 161
- save() (nipyype.pipeline.engine.workflows.Workflow method), 166
- save_inputs_to_json() (nipyype.interfaces.base.BaseInterface method), 63
- save_inputs_to_json() (nipyype.interfaces.base.CommandLine method), 83
- save_inputs_to_json() (nipyype.interfaces.base.MpiCommandLine method), 105
- save_inputs_to_json() (nipyype.interfaces.base.SEMLikeCommandLine method), 121
- save_inputs_to_json() (nipyype.interfaces.base.StdOutCommandLine method), 123
- save_inputs_to_json() (nipyype.interfaces.matlab.MatlabCommand method), 145
- scalartransform, 1417
- Scrubmask, 742
- Seg, 627
- Segment, 1587
- SegmentCC, 925
- SegmentWM, 926
- SegStats, 884
- SegStatsReconAll, 888
- Select, 1604
- SelectFiles, 1636
- SEMLikeCommandLine (class in nipyype.interfaces.base), 121
- set() (nipyype.interfaces.base.BaseInterfaceInputSpec method), 70
- set() (nipyype.interfaces.base.BaseTraitedSpec method), 78
- set() (nipyype.interfaces.base.Bunch method), 81
- set() (nipyype.interfaces.base.CommandLineInputSpec method), 89
- set() (nipyype.interfaces.base.DynamicTraitedSpec method), 98
- set() (nipyype.interfaces.base.MpiCommandLineInputSpec method), 112
- set() (nipyype.interfaces.base.OutputMultiPath method), 120

set() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 129
 set() (nipy.interfaces.base.TraitedSpec method), 141
 set() (nipy.interfaces.matlab.MatlabInputSpec method), 152
 set_default_matlab_cmd() (nipy.interfaces.matlab.MatlabCommand class method), 145
 set_default_mfile() (nipy.interfaces.matlab.MatlabCommand class method), 145
 set_default_paths() (nipy.interfaces.matlab.MatlabCommand class method), 145
 set_default_terminal_output() (nipy.interfaces.base.CommandLine class method), 83
 set_default_terminal_output() (nipy.interfaces.base.MpiCommandLine method), 105
 set_default_terminal_output() (nipy.interfaces.base.SEMLikeCommandLine method), 122
 set_default_terminal_output() (nipy.interfaces.base.StdOutCommandLine method), 123
 set_default_terminal_output() (nipy.interfaces.matlab.MatlabCommand method), 145
 set_input() (nipy.pipeline.engine.nodes.JoinNode method), 157
 set_input() (nipy.pipeline.engine.nodes.MapNode method), 159
 set_input() (nipy.pipeline.engine.nodes.Node method), 161
 set_trait_dispatch_handler() (nipy.interfaces.base.BaseInterfaceInputSpec method), 70
 set_trait_dispatch_handler() (nipy.interfaces.base.BaseTraitedSpec method), 79
 set_trait_dispatch_handler() (nipy.interfaces.base.CommandLineInputSpec method), 90
 set_trait_dispatch_handler() (nipy.interfaces.base.DynamicTraitedSpec method), 98
 set_trait_dispatch_handler() (nipy.interfaces.base.MpiCommandLineInputSpec method), 112
 set_trait_dispatch_handler() (nipy.interfaces.base.StdOutCommandLineInputSpec method), 129
 set_trait_dispatch_handler() (nipy.interfaces.base.TraitedSpec method), 141
 set_trait_dispatch_handler() (nipy.interfaces.matlab.MatlabInputSpec method), 152
 set_value() (nipy.interfaces.base.InputMultiPath method), 103
 set_value() (nipy.interfaces.base.MultiPath method), 117
 set_value() (nipy.interfaces.base.OutputMultiPath method), 120
 set_value() (nipy.interfaces.base.Str method), 134
 setup() (in module nipy.sphinxext.plot_workflow), 169
 SFLUTGen, 747
 SFPeaks, 806
 SFPICOCalibData, 749
 Shredder, 761
 ShuffleVectorsModule, 1470
 SigLoss, 1021, 1125
 SignalExtraction, 1645
 Similarity, 568, 1336
 SimilarityIndex, 1344
 SimpleRegionGrowingSegmentation, 1548
 SimpleThreshold, 580
 SimulateMultiTensor, 847
 Skullfinder, 743
 SkullStrip, 628
 Slicer, 1126
 SlicerCommandLine, 1475, 1623
 SliceTimer, 1100
 SliceTiming, 1589
 SMM, 1070
 Smooth, 927, 1128, 1590
 SmoothEstimate, 1071
 SmoothTessellation, 973
 SpaceTimeRealigner, 1334
 SpatialFilter, 1044
 SpecifyModel, 585
 SpecifySparseModel, 588
 SpecifySPMModel, 587
 Sphere, 975
 SphericalAverage, 892
 SphericallyDeconvolutedStreamlineTrack, 1245
 SplineFilter, 837
 Split, 1129, 1605
 SplitNifti, 1620
 SplitROIs, 580
 SQLiteSink, 1634
 SSHDataGrabber, 1635
 STAPLEAnalysis, 1415
 StatsCommand, 1326
 Storage, 1045
 StdOutCommandLine (class in nipy.interfaces.base), 122
 StdOutCommandLineInputSpec (class in nipy.interfaces.base), 123

StimulusCorrelation, 595
 Str (class in nipyre.interfaces.base), 131
 Stream (class in nipyre.interfaces.base), 134
 StreamlineTrack, 1247
 StreamlineTractography, 850
 SubtractScalarVolumes, 1493
 Surface2VolTransform, 976
 SurfaceSmooth, 977
 SurfaceSnapshots, 979
 SurfaceTransform, 981
 SUSAN, 1099
 SVMTest, 638
 SVMTrain, 639
 SVReg, 739
 SwapDimensions, 1130
 sync_trait() (nipyre.interfaces.base.BaseInterfaceInputSpec method), 70
 sync_trait() (nipyre.interfaces.base.BaseTraitedSpec method), 79
 sync_trait() (nipyre.interfaces.base.CommandLineInputSpec method), 90
 sync_trait() (nipyre.interfaces.base.DynamicTraitedSpec method), 98
 sync_trait() (nipyre.interfaces.base.MpiCommandLineInputSpec method), 112
 sync_trait() (nipyre.interfaces.base.StdOutCommandLineInputSpec method), 129
 sync_trait() (nipyre.interfaces.base.TraitedSpec method), 141
 sync_trait() (nipyre.interfaces.matlab.MatlabInputSpec method), 152
 synchronize_iterables() (in module nipyre.pipeline.engine.utils), 163
 SynthesizeFLASH, 928

T

TalairachAVI, 983
 TalairachQC, 984
 Tca, 744
 TCat, 657
 TCK2VTK, 1271
 TCompCor, 555
 TCorr1D, 629
 TCorrelate, 633
 TCorrMap, 630
 TemporalFilter, 1046
 Tensor2ApparentDiffusion, 1226
 Tensor2FractionalAnisotropy, 1227
 Tensor2Vector, 1228
 TensorMetrics, 1272
 TensorMode, 849
 TextureFromNoiseImageFilter, 1415
 TextureMeasureFilter, 1416
 ThicknessPVC, 745
 Threshold, 1047, 1229, 1572
 ThresholdScalarVolume, 1508
 ThresholdStatistics, 1573
 Tkregister2, 985
 To3D, 659
 ToEcat, 1175
 topological_sort() (in module nipyre.pipeline.engine.utils), 163
 TOPUP, 1022
 ToRaw, 1176
 Track, 778
 TrackBallStick, 781
 TrackBayesDirac, 783
 TrackBedpostxDeter, 787
 TrackBedpostxProba, 789
 TrackBootstrap, 792
 TrackDensityMap, 851
 TrackDT, 795
 TrackMerge, 838
 TrackPICO, 798
 Tracks2Prob, 1250
 Trackvis2Camino, 812
 Tractography, 1262
 TractographyLabelMapSeeding, 1487
 TractShredder, 763
 TractSkeleton, 1006
 Training, 1027
 TrainingSetCreator, 1028
 trait() (nipyre.interfaces.base.BaseInterfaceInputSpec method), 70
 trait() (nipyre.interfaces.base.BaseTraitedSpec method), 79
 trait() (nipyre.interfaces.base.CommandLineInputSpec method), 90
 trait() (nipyre.interfaces.base.DynamicTraitedSpec method), 98
 trait() (nipyre.interfaces.base.MpiCommandLineInputSpec method), 113
 trait() (nipyre.interfaces.base.StdOutCommandLineInputSpec method), 130
 trait() (nipyre.interfaces.base.TraitedSpec method), 141
 trait() (nipyre.interfaces.matlab.MatlabInputSpec method), 153
 trait_context() (nipyre.interfaces.base.BaseInterfaceInputSpec method), 70
 trait_context() (nipyre.interfaces.base.BaseTraitedSpec method), 79
 trait_context() (nipyre.interfaces.base.CommandLineInputSpec method), 90
 trait_context() (nipyre.interfaces.base.DynamicTraitedSpec method), 99
 trait_context() (nipyre.interfaces.base.MpiCommandLineInputSpec method), 113

trait_context() (nipyne.interfaces.base.StdOutCommandLineInputSpec method), 130	trait_set() (nipyne.interfaces.base.BaseInterfaceInputSpec method), 71
trait_context() (nipyne.interfaces.base.TraitSpec method), 142	trait_names() (nipyne.interfaces.base.BaseTraitSpec method), 79
trait_context() (nipyne.interfaces.matlab.MatlabInputSpec method), 153	trait_names() (nipyne.interfaces.base.CommandLineInputSpec method), 90
trait_get() (nipyne.interfaces.base.BaseInterfaceInputSpec method), 70	trait_names() (nipyne.interfaces.base.DynamicTraitSpec method), 99
trait_get() (nipyne.interfaces.base.BaseTraitSpec method), 79	trait_names() (nipyne.interfaces.base.MpiCommandLineInputSpec method), 113
trait_get() (nipyne.interfaces.base.CommandLineInputSpec method), 90	trait_names() (nipyne.interfaces.base.StdOutCommandLineInputSpec method), 130
trait_get() (nipyne.interfaces.base.DynamicTraitSpec method), 99	trait_names() (nipyne.interfaces.base.TraitSpec method), 142
trait_get() (nipyne.interfaces.base.MpiCommandLineInputSpec method), 113	trait_names() (nipyne.interfaces.matlab.MatlabInputSpec method), 153
trait_get() (nipyne.interfaces.base.StdOutCommandLineInputSpec method), 130	trait_property_changed() (nipyne.interfaces.base.BaseInterfaceInputSpec method), 71
trait_get() (nipyne.interfaces.base.TraitSpec method), 142	trait_property_changed() (nipyne.interfaces.base.BaseTraitSpec method), 79
trait_get() (nipyne.interfaces.matlab.MatlabInputSpec method), 153	trait_property_changed() (nipyne.interfaces.base.CommandLineInputSpec method), 90
trait_items_event() (nipyne.interfaces.base.BaseInterfaceInputSpec method), 70	trait_property_changed() (nipyne.interfaces.base.DynamicTraitSpec method), 99
trait_items_event() (nipyne.interfaces.base.BaseTraitSpec method), 79	trait_property_changed() (nipyne.interfaces.base.MpiCommandLineInputSpec method), 113
trait_items_event() (nipyne.interfaces.base.CommandLineInputSpec method), 90	trait_property_changed() (nipyne.interfaces.base.StdOutCommandLineInputSpec method), 130
trait_items_event() (nipyne.interfaces.base.DynamicTraitSpec method), 99	trait_property_changed() (nipyne.interfaces.base.TraitSpec method), 142
trait_items_event() (nipyne.interfaces.base.MpiCommandLineInputSpec method), 113	trait_property_changed() (nipyne.interfaces.matlab.MatlabInputSpec method), 153
trait_items_event() (nipyne.interfaces.base.StdOutCommandLineInputSpec method), 130	trait_set() (nipyne.interfaces.base.BaseInterfaceInputSpec method), 71
trait_items_event() (nipyne.interfaces.base.TraitSpec method), 142	trait_set() (nipyne.interfaces.base.BaseTraitSpec method), 79
trait_items_event() (nipyne.interfaces.matlab.MatlabInputSpec method), 153	trait_set() (nipyne.interfaces.base.CommandLineInputSpec method), 90
trait_monitor() (nipyne.interfaces.base.BaseInterfaceInputSpec method), 70	trait_set() (nipyne.interfaces.base.DynamicTraitSpec method), 99
trait_monitor() (nipyne.interfaces.base.BaseTraitSpec method), 79	trait_set() (nipyne.interfaces.base.MpiCommandLineInputSpec method), 113
trait_monitor() (nipyne.interfaces.base.CommandLineInputSpec method), 90	trait_set() (nipyne.interfaces.base.StdOutCommandLineInputSpec method), 130
trait_monitor() (nipyne.interfaces.base.DynamicTraitSpec method), 99	trait_set() (nipyne.interfaces.base.TraitSpec method), 142
trait_monitor() (nipyne.interfaces.base.MpiCommandLineInputSpec method), 113	trait_set() (nipyne.interfaces.matlab.MatlabInputSpec method), 153
trait_monitor() (nipyne.interfaces.base.StdOutCommandLineInputSpec method), 130	
trait_monitor() (nipyne.interfaces.base.TraitSpec method), 142	
trait_monitor() (nipyne.interfaces.matlab.MatlabInputSpec method), 153	

1677

[traits_init\(\) \(nipy.interfaces.base.StdOutCommandLineInputSpec method\), 131](#)
[traits_init\(\) \(nipy.interfaces.base.TraitedSpec method\), 143](#)
[traits_init\(\) \(nipy.interfaces.matlab.MatlabInputSpec method\), 154](#)
[traits_inited\(\) \(nipy.interfaces.base.BaseInterfaceInputSpec method\), 72](#)
[traits_inited\(\) \(nipy.interfaces.base.BaseTraitedSpec method\), 80](#)
[traits_inited\(\) \(nipy.interfaces.base.CommandLineInputSpec method\), 91](#)
[traits_inited\(\) \(nipy.interfaces.base.DynamicTraitedSpec method\), 100](#)
[traits_inited\(\) \(nipy.interfaces.base.MpiCommandLineInputSpec method\), 114](#)
[traits_inited\(\) \(nipy.interfaces.base.StdOutCommandLineInputSpec method\), 131](#)
[traits_inited\(\) \(nipy.interfaces.base.TraitedSpec method\), 143](#)
[traits_inited\(\) \(nipy.interfaces.matlab.MatlabInputSpec method\), 154](#)
[Trim, 1335](#)
[TShift, 634](#)
[TSNR, 556, 581](#)
[TStat, 658](#)
[TupleMaths, 1319](#)
[TVTKBaseInterface, 563](#)
[TwoSampleTTestDesign, 1574](#)

U

[UKFTractography, 1396](#)
[UnaryMaths, 1048, 1321](#)
[UnaryStats, 1327](#)
[UnbiasedNonLocalMeans, 1400](#)
[unescape_doctest\(\) \(in module nipy.sphinxext.plot_workflow\), 170](#)
[Unifize, 661](#)
[UnpackSDICOMDir, 929](#)
[update\(\) \(nipy.interfaces.base.Bunch method\), 81](#)
[update\(\) \(nipy.pipeline.engine.nodes.JoinNode method\), 158](#)
[update\(\) \(nipy.pipeline.engine.nodes.MapNode method\), 159](#)
[update\(\) \(nipy.pipeline.engine.nodes.Node method\), 161](#)

V

[validate\(\) \(nipy.interfaces.base.InputMultiPath method\), 103](#)
[validate\(\) \(nipy.interfaces.base.MultiPath method\), 117](#)
[validate\(\) \(nipy.interfaces.base.OutputMultiPath method\), 120](#)
[validate\(\) \(nipy.interfaces.base.Str method\), 134](#)
[validate_trait\(\) \(nipy.interfaces.base.BaseInterfaceInputSpec method\), 72](#)
[validate_trait\(\) \(nipy.interfaces.base.BaseTraitedSpec method\), 80](#)
[validate_trait\(\) \(nipy.interfaces.base.CommandLineInputSpec method\), 91](#)
[validate_trait\(\) \(nipy.interfaces.base.DynamicTraitedSpec method\), 100](#)
[validate_trait\(\) \(nipy.interfaces.base.MpiCommandLineInputSpec method\), 114](#)
[validate_trait\(\) \(nipy.interfaces.base.StdOutCommandLineInputSpec method\), 131](#)
[validate_trait\(\) \(nipy.interfaces.base.TraitedSpec method\), 143](#)
[validate_trait\(\) \(nipy.interfaces.matlab.MatlabInputSpec method\), 154](#)
[VBRAINSDemonWarp, 1433, 1544](#)
[VecReg, 1008](#)
[version \(nipy.interfaces.base.BaseInterface attribute\), 63](#)
[version \(nipy.interfaces.base.CommandLine attribute\), 83](#)
[version \(nipy.interfaces.base.Interface attribute\), 104](#)
[version \(nipy.interfaces.base.MpiCommandLine attribute\), 106](#)
[version \(nipy.interfaces.base.SEMLikeCommandLine attribute\), 122](#)
[version \(nipy.interfaces.base.StdOutCommandLine attribute\), 123](#)
[version \(nipy.interfaces.matlab.MatlabCommand attribute\), 146](#)
[version_from_command\(\) \(nipy.interfaces.base.CommandLine method\), 83](#)
[version_from_command\(\) \(nipy.interfaces.base.MpiCommandLine method\), 106](#)
[version_from_command\(\) \(nipy.interfaces.base.SEMLikeCommandLine method\), 122](#)
[version_from_command\(\) \(nipy.interfaces.base.StdOutCommandLine method\), 123](#)
[version_from_command\(\) \(nipy.interfaces.matlab.MatlabCommand method\), 146](#)
[visible_traits\(\) \(nipy.interfaces.base.BaseInterfaceInputSpec method\), 72](#)
[visible_traits\(\) \(nipy.interfaces.base.BaseTraitedSpec method\), 80](#)
[visible_traits\(\) \(nipy.interfaces.base.CommandLineInputSpec method\), 91](#)
[visible_traits\(\) \(nipy.interfaces.base.DynamicTraitedSpec method\), 100](#)

- method), 100
 - visible_traits() (nipy.interfaces.base.MpiCommandLineInputSpec attribute), 143
 - method), 114
 - visible_traits() (nipy.interfaces.base.StdOutCommandLineInputSpec attribute), 154
 - method), 131
 - visible_traits() (nipy.interfaces.base.TraitedSpec attribute), 143
 - method), 143
 - visible_traits() (nipy.interfaces.matlab.MatlabInputSpec attribute), 154
 - method), 154
 - Vnifti2Image, 1607
 - Volcentre, 1179
 - Voliso, 1180
 - Volpad, 1181
 - Volreg, 635
 - VolSymm, 1178
 - VolumeMask, 986
 - VotingBinaryHoleFillingImageFilter, 1510
 - VtkStreamlines, 764
 - VtoMat, 1608
- ## W
- walk() (in module nipy.pipeline.engine.utils), 163
 - walk_files() (in module nipy.pipeline.engine.utils), 163
 - walk_outputs() (in module nipy.pipeline.engine.utils), 163
 - Warp, 637
 - WarpImageMultiTransform, 681
 - WarpPoints, 563, 1131
 - WarpPointsToStd, 1132
 - WarpTimeSeriesImageMultiTransform, 683
 - WarpUtils, 1134
 - WatershedBEM, 1215
 - WatershedSkullStrip, 931
 - wf_directive() (in module nipy.sphinxext.plot_workflow), 170
 - with_traceback() (nipy.interfaces.base.NipyInterfaceError method), 117
 - with_traceback() (nipy.sphinxext.plot_workflow.GraphError method), 169
 - Workflow (class in nipy.pipeline.engine.workflows), 164
 - wrappers (nipy.interfaces.base.BaseInterfaceInputSpec attribute), 72
 - wrappers (nipy.interfaces.base.BaseTraitedSpec attribute), 80
 - wrappers (nipy.interfaces.base.CommandLineInputSpec attribute), 91
 - wrappers (nipy.interfaces.base.DynamicTraitedSpec attribute), 100
 - wrappers (nipy.interfaces.base.MpiCommandLineInputSpec attribute), 114
 - wrappers (nipy.interfaces.base.StdOutCommandLineInputSpec attribute), 131
 - wrappers (nipy.interfaces.base.TraitedSpec attribute), 154
 - wrappers (nipy.interfaces.matlab.MatlabInputSpec attribute), 154
 - write_graph() (nipy.pipeline.engine.workflows.Workflow method), 166
 - write_hierarchical_dotfile() (nipy.pipeline.engine.workflows.Workflow method), 166
 - write_report() (nipy.pipeline.engine.nodes.JoinNode method), 158
 - write_report() (nipy.pipeline.engine.nodes.MapNode method), 159
 - write_report() (nipy.pipeline.engine.nodes.Node method), 161
 - write_workflow_prov() (in module nipy.pipeline.engine.utils), 163
- ## X
- XFibres5, 1009
 - XfmAvg, 1183
 - XfmConcat, 1184
 - XfmInvert, 1185
 - XNATSink, 1637
 - XNATSource, 1638
- ## Z
- ZCutUp, 662