
nipype Documentation

Release 1.1.7

Neuroimaging in Python team

December 17, 2018, 15:27 PDT

Contents

1	workflows.data	3
2	workflows.dmri	5
3	workflows.fmri	81
4	workflows.misc	105
5	workflows.rsfmri	107
6	workflows.smri	113
7	dMRI: Camino, DTI	133
8	dMRI: Connectivity - Camino, CMTK, FreeSurfer	139
9	dMRI: Connectivity - MRtrix, CMTK, FreeSurfer	149
10	dMRI: DTI - Diffusion Toolkit, FSL	161
11	dMRI: HARDI - Diffusion Toolkit, FSL	165
12	dMRI: DTI, FSL	169
13	dMRI: Group connectivity - Camino, FSL, FreeSurfer	173
14	dMRI: Group connectivity - MRtrix, FSL, FreeSurfer	177
15	dMRI: DTI - MRtrix, FSL	181
16	dMRI: Preprocessing	187
17	dMRI: TBSS on NKI RS data	191
18	fMRI: OpenfMRI.org data, FSL, ANTS, c3daffine	195
19	fMRI: surface smooth - FreeSurfer, SPM	215
20	fMRI: FSL	225

21 fMRI: FEEDS - FSL	235
22 fMRI: FSL reuse workflows	239
23 fMRI: NiPy GLM, SPM	245
24 fMRI: Coregistration - Slicer, BRAINS	251
25 fMRI: SPM, FSL	255
26 fMRI: SPM Auditory dataset	261
27 fMRI: DARTEL, SPM	267
28 fMRI: Famous vs non-famous faces, SPM	277
29 fMRI: SPM nested workflows	285
30 HOWTO: Using caching without using Workflow	293
31 rsfMRI: ANTS, FS, FSL, SPM, aCompCor	295
32 rsfMRI: ANTS, FS, FSL, NiPy, aCompCor	315
33 Paper: Smoothing comparison	335
34 sMRI: Using new ANTS for creating a T1 template	339
35 sMRI: Using ANTS for registration	343
36 sMRI: Using new ANTS for creating a T1 template (ITK4)	345
37 sMRI: USing CBS Tools for skullstripping	349
38 sMRI: FreeSurfer	351
39 sMRI: FSReconAll	353
40 sMRI: Regional Tessellation and Surface Smoothing	355
41 Workflow from scratch	359
42 Workshop: Dartmouth College 2010	363
43 algorithms.confounds	369
44 algorithms.icc	379
45 algorithms.mesh	381
46 algorithms.metrics	385
47 algorithms.misc	391
48 algorithms.modelgen	405
49 algorithms.rapidart	413
50 algorithms.stats	417

51	interfaces.afni	419
52	interfaces.ants	557
53	interfaces.base	609
54	interfaces.brainsuite	615
55	interfaces.camino	645
56	interfaces.camino2trackvis	705
57	interfaces.cmtk	709
58	interfaces.diffusion_toolkit	721
59	interfaces.dipy	731
60	interfaces.dtitk	743
61	interfaces.elastix	769
62	interfaces.freesurfer	775
63	interfaces.fsl	903
64	interfaces.minc	1049
65	interfaces.mipav	1097
66	interfaces.mixins	1123
67	interfaces.mne	1125
68	interfaces.mrtrix	1127
69	interfaces.mrtrix3	1159
70	interfaces.niftyfit	1187
71	interfaces.niftyreg	1201
72	interfaces.niftyseg	1217
73	interfaces.nipy	1239
74	interfaces.nitime	1245
75	interfaces.semtools	1247
76	interfaces.slicer	1367
77	interfaces.spm	1447
78	interfaces.utility	1489
79	interfaces.vista	1495
80	interfaces.workbench	1497

81	interfaces.bru2nii	1501
82	interfaces.c3	1503
83	interfaces.dcm2nii	1507
84	interfaces.dcmstack	1511
85	interfaces.dynamic_slicer	1515
86	interfaces.image	1517
87	interfaces.io	1521
88	interfaces.meshfix	1537
89	interfaces.nilearn	1541
90	interfaces.petpvc	1543
91	interfaces.quickshear	1547
92	interfaces.vtkbase	1549
93	API	1551
94	Developer Guide	1583
	Bibliography	1615
	Python Module Index	1617

Previous versions: [1.1.6](#) [1.1.5](#)

Michael Notter's Nipype guide

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

Interfaces, Workflows and Examples

- [Workflows](#)

1.1 `get_flirt_schedule()`

[Link to code](#)

2.1 workflows.dmri.camino.connectivity_mapping

2.1.1 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
```

(continues on next page)

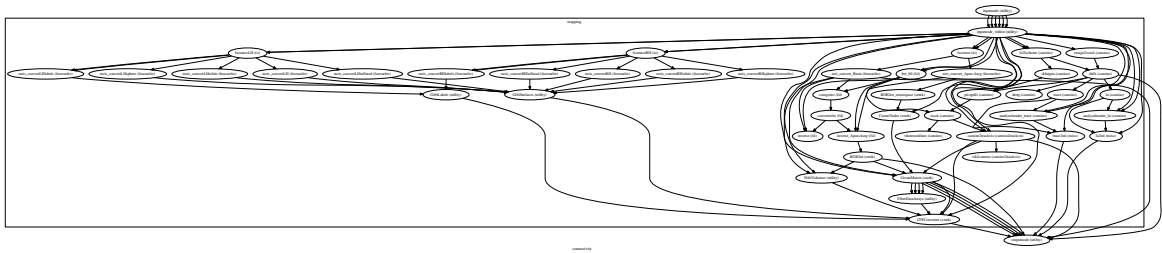
(continued from previous page)

```

outputnode.fa
outputnode.struct
outputnode.trace
outputnode.tracts
outputnode.tensors

```

Graph



2.2 workflows.dmri.camino.diffusion

2.2.1 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

```

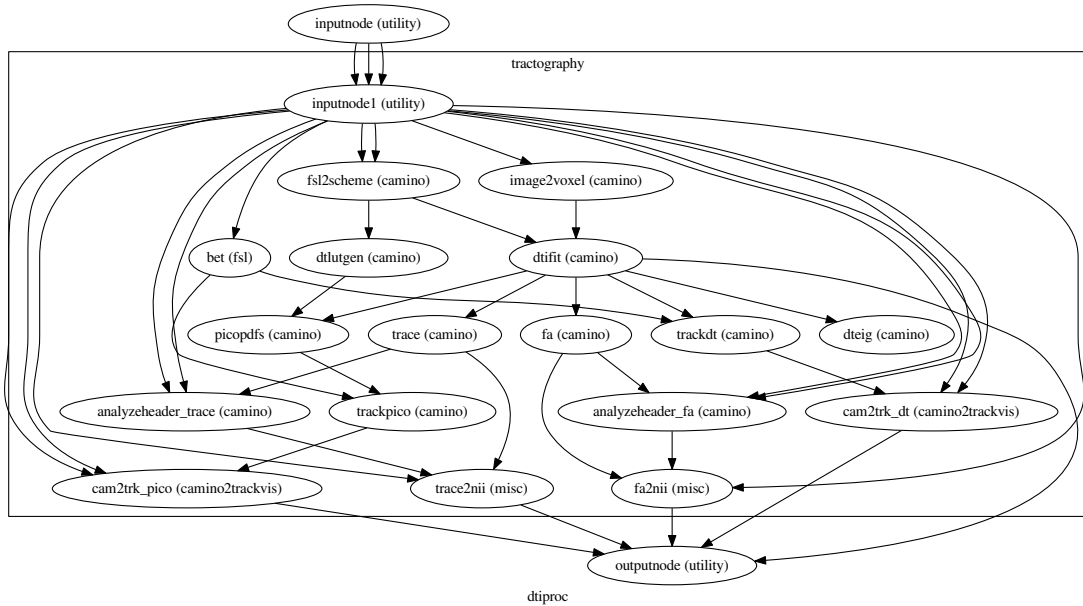
Outputs:

```

outputnode.fa
outputnode.trace
outputnode.tracts_pico
outputnode.tracts_dt
outputnode.tensors

```

Graph



2.3 workflows.dmri.camino.group_connectivity

2.3.1 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']])
```

2.4 workflows.dmri.connectivity.group_connectivity

2.4.1 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.

2.4.2 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')
```

2.4.3 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')
```

2.4.4 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')
```

2.4.5 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
```

2.4.6 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
```

2.4.7 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

2.5 workflows.dmri.connectivity.nx

2.5.1 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 nipyype.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()
```

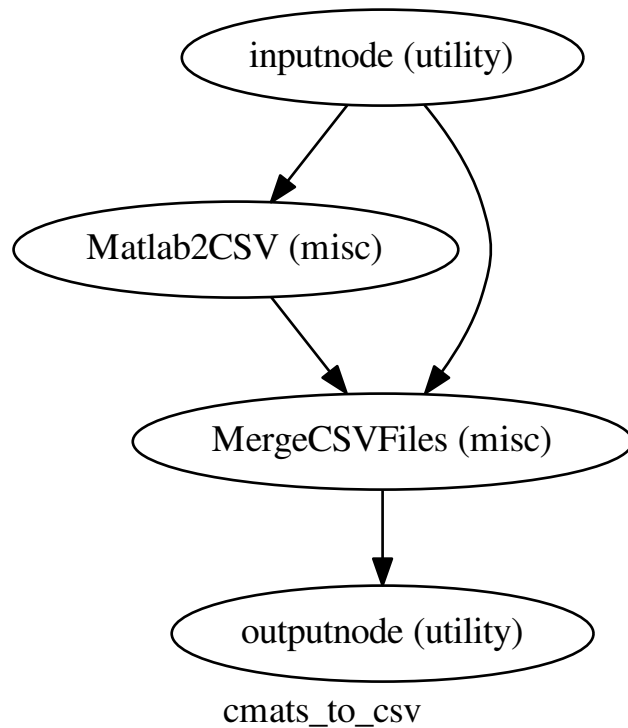
Inputs:

```
inputnode.extra_field
inputnode.matlab_matrix_files
```

Outputs:

```
outputnode.csv_file
```

Graph



2.5.2 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
  
```

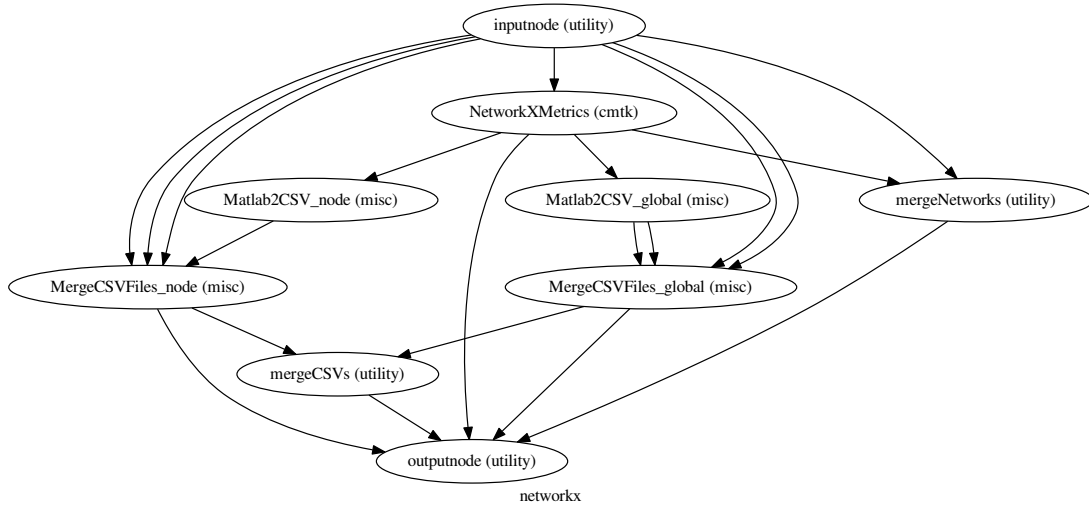
Outputs:

```

outputnode.network_files
outputnode.csv_files
outputnode.matlab_files

```

Graph



2.5.3 add_global_to_filename()

[Link to code](#)

2.5.4 add_nodal_to_filename()

[Link to code](#)

2.6 workflows.dmri.dipy.denoise

2.6.1 nlmeans_pipeline()

[Link to code](#)

Workflow that performs nlmeans denoising

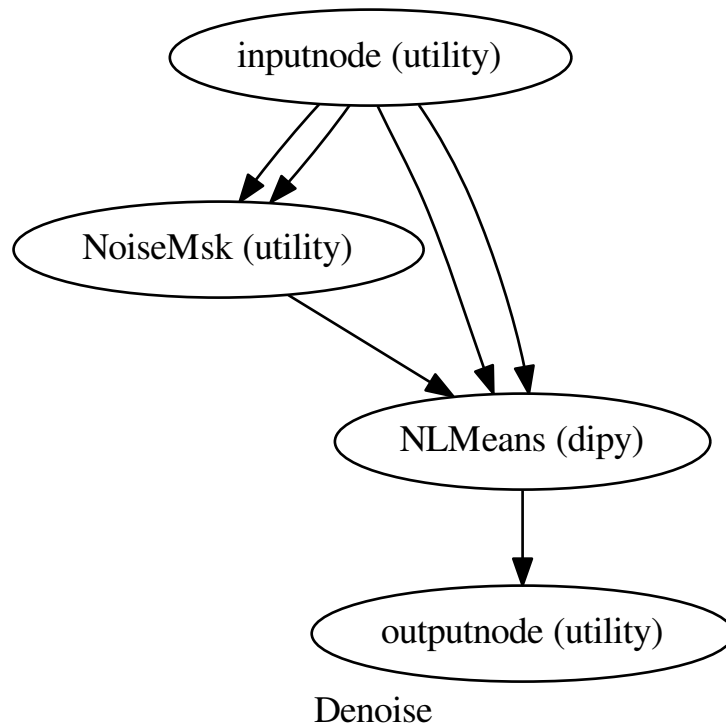
Example

```

>>> 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



2.6.2 `bg_mask()`

[Link to code](#)

Rough mask of background from brain masks

2.6.3 `csf_mask()`

[Link to code](#)

Artesanal mask of csf in T2w-like images

2.7 `workflows.dmri.dtitk.tensor_registration`

2.7.1 `affine_tensor_pipeline()`

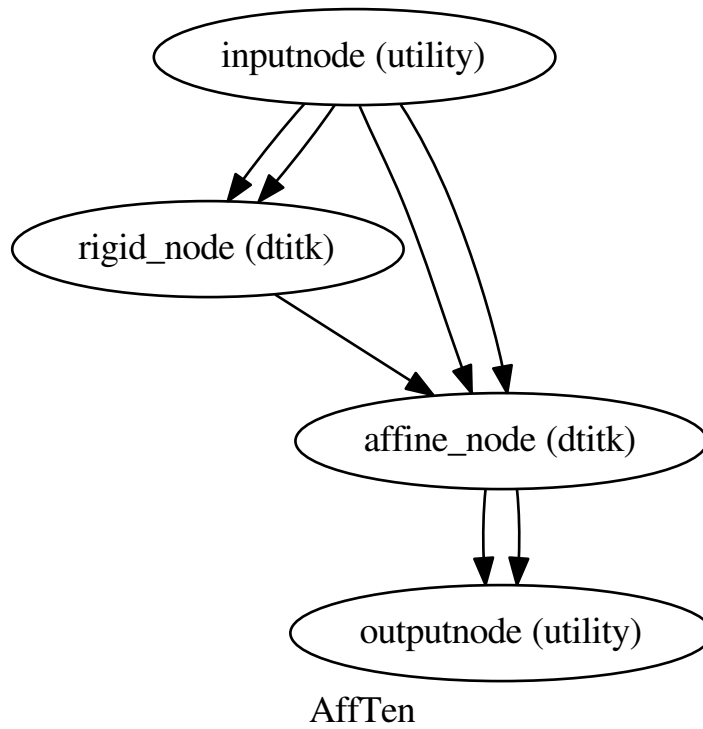
[Link to code](#)

Workflow that performs a linear registration (Rigid followed by Affine)

Example

```
>>> from nipy.workflows.dmri.dtitk.tensor_registration import affine_tensor_
      ↪pipeline
>>> affine = affine_tensor_pipeline()
>>> affine.inputs.inputnode.fixed_file = 'im1.nii'
>>> affine.inputs.inputnode.moving_file = 'im2.nii'
>>> affine.run()
```

Graph



2.7.2 diffeomorphic_tensor_pipeline()

[Link to code](#)

Workflow that performs a diffeomorphic registration (Rigid and Affine followed by Diffeomorphic) Note: the requirements for a diffeomorphic registration specify that the dimension 0 is a power of 2 so images are resliced prior to registration. Remember to move origin and reslice prior to applying xfm to another file!

Example

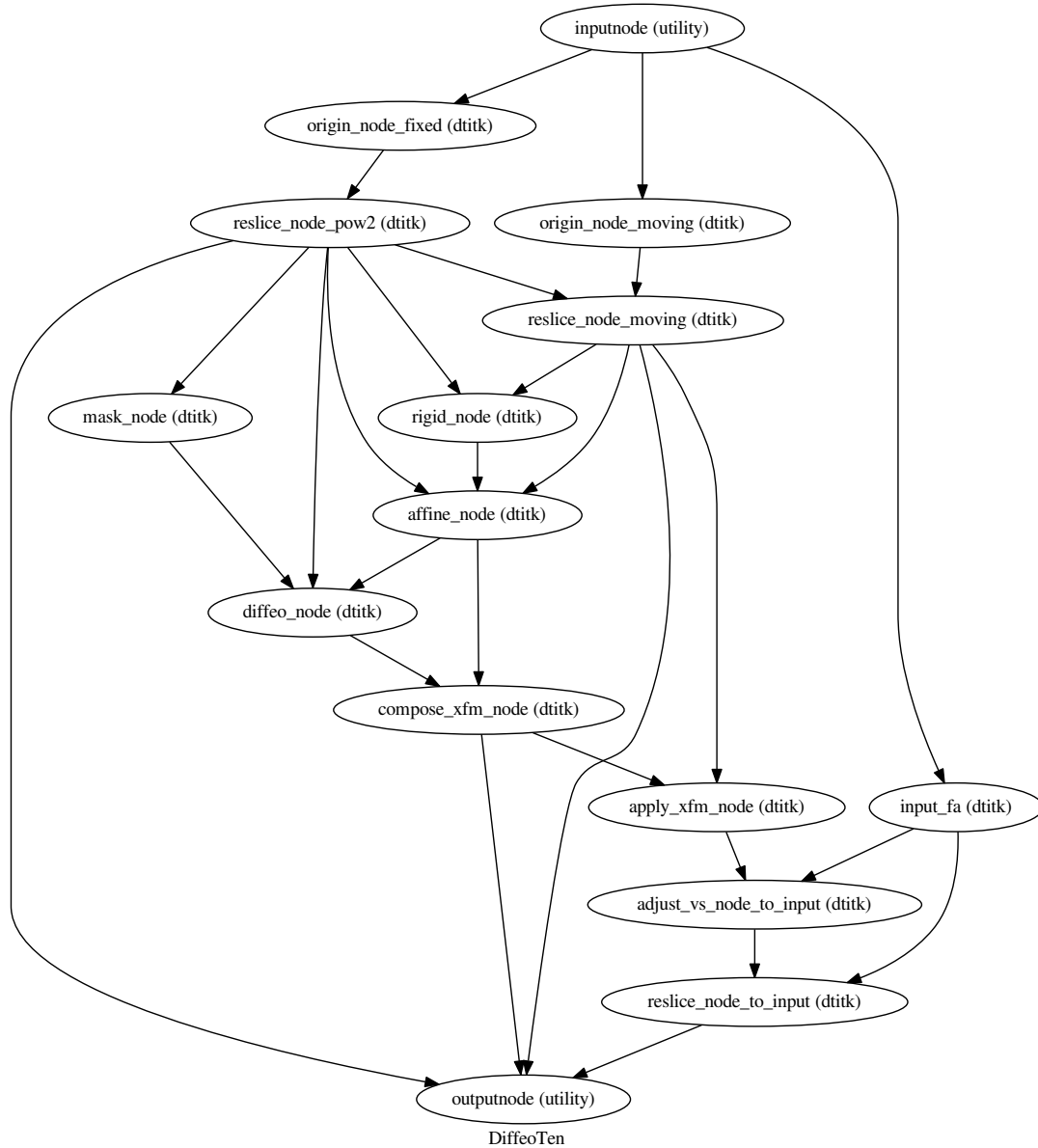
```
>>> from nipy.workflows.dmri.dtitk.tensor_registration import diffeomorphic_
      ↪tensor_pipeline
>>> diffeo = diffeomorphic_tensor_pipeline()
>>> diffeo.inputs.inputnode.fixed_file = 'im1.nii'
```

(continues on next page)

(continued from previous page)

```
>>> diffeo.inputs.inputnode.moving_file = 'im2.nii'
>>> diffeo.run()
```

Graph



2.8 workflows.dmri.fsl.artifacts

2.8.1 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).

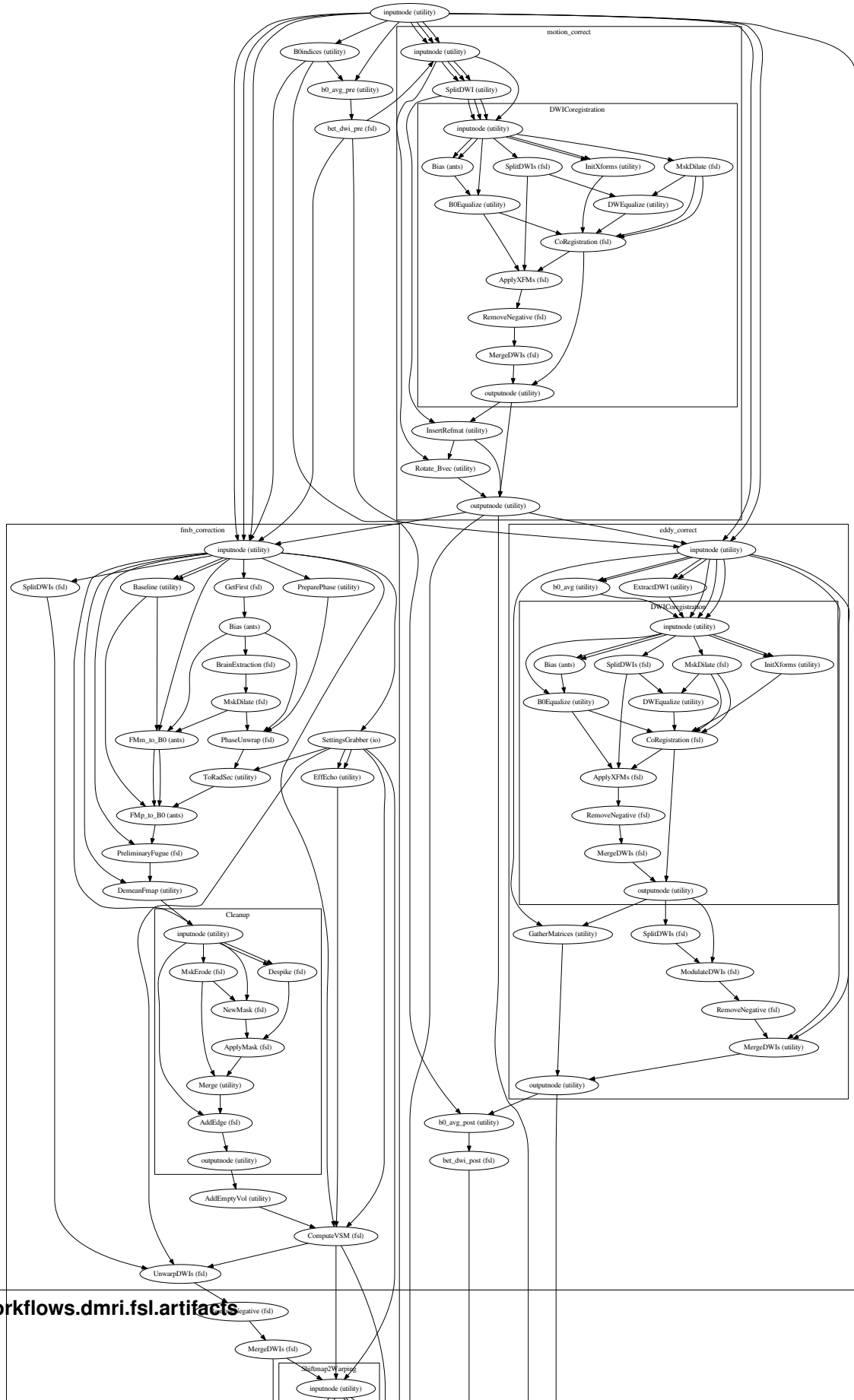
The displacement fields from each kind of distortions are combined. Thus, only one interpolation occurs between input data and result.

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



2.8.2 all_fsl_pipeline()

[Link to code](#)

Workflow that integrates FSL `topup` and `eddy`.

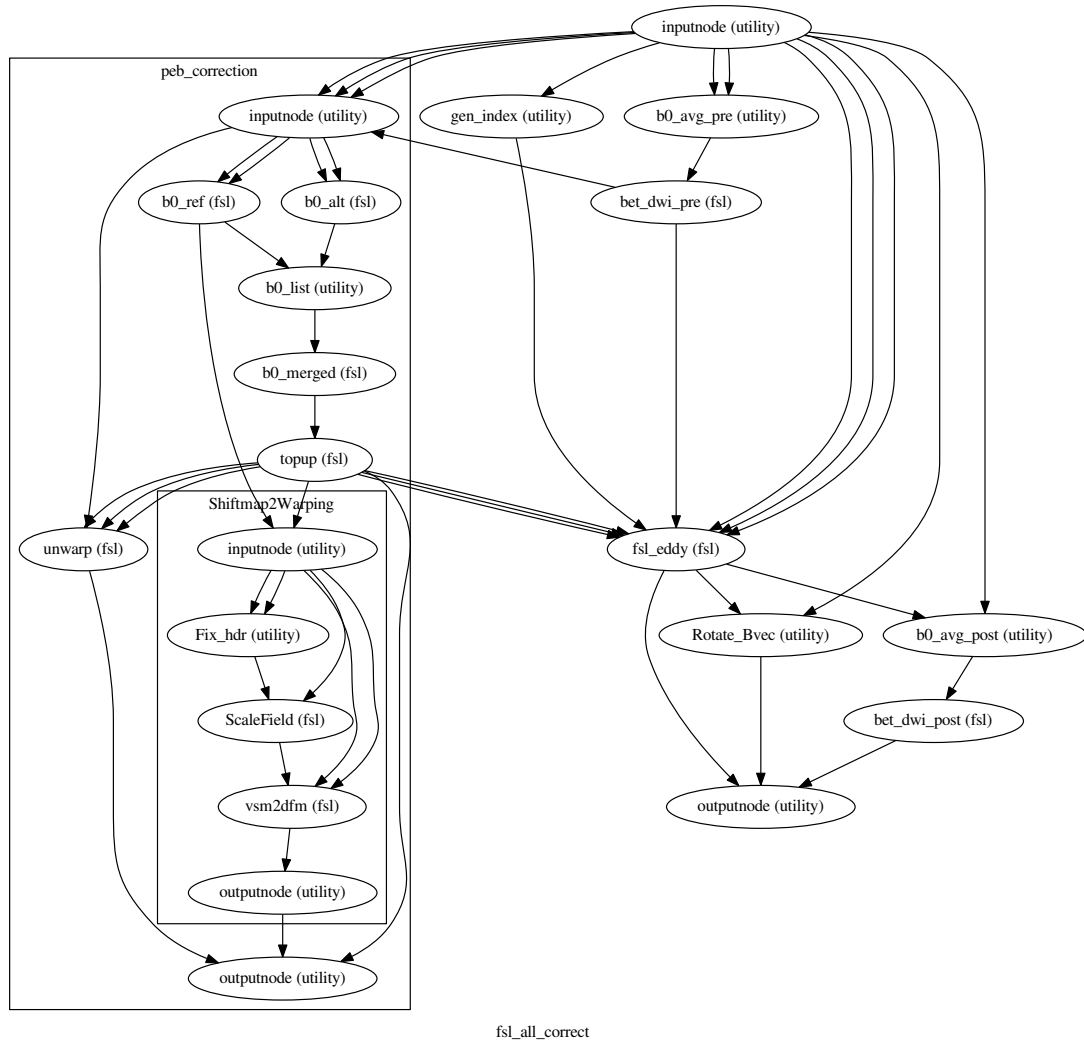
Warning: this workflow rotates the gradients table (*b*-vectors) [*Leemans09*].

Warning: this workflow does not perform jacobian modulation of each *DWI* [*Jones10*].

Examples

```
>>> from nipy.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



2.8.3 all_peg_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

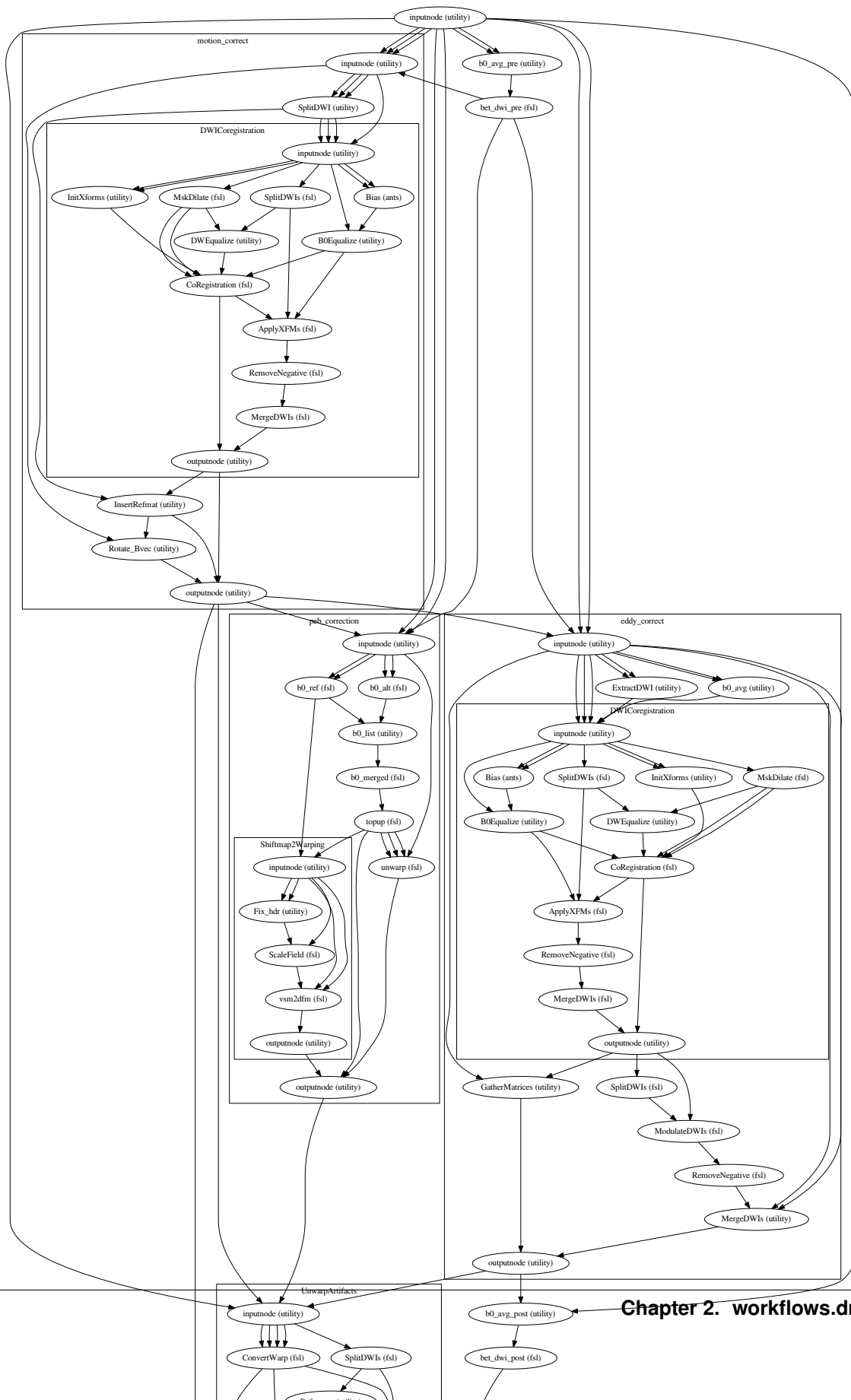
```
>>> from nipyype.workflows.dmri.fsl.artifacts import all_peg_pipeline
>>> allcorr = all_peg_pipeline()
>>> allcorr.inputs.inputnode.in_file = 'epi.nii'
>>> allcorr.inputs.inputnode.alt_file = 'epi_rev.nii'
```

(continues on next page)

(continued from previous page)

```
>>> allcorr.inputs.inputnode.in_bval = 'diffusion.bval'
>>> allcorr.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> allcorr.run()
```


Graph



2.8.4 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()
```

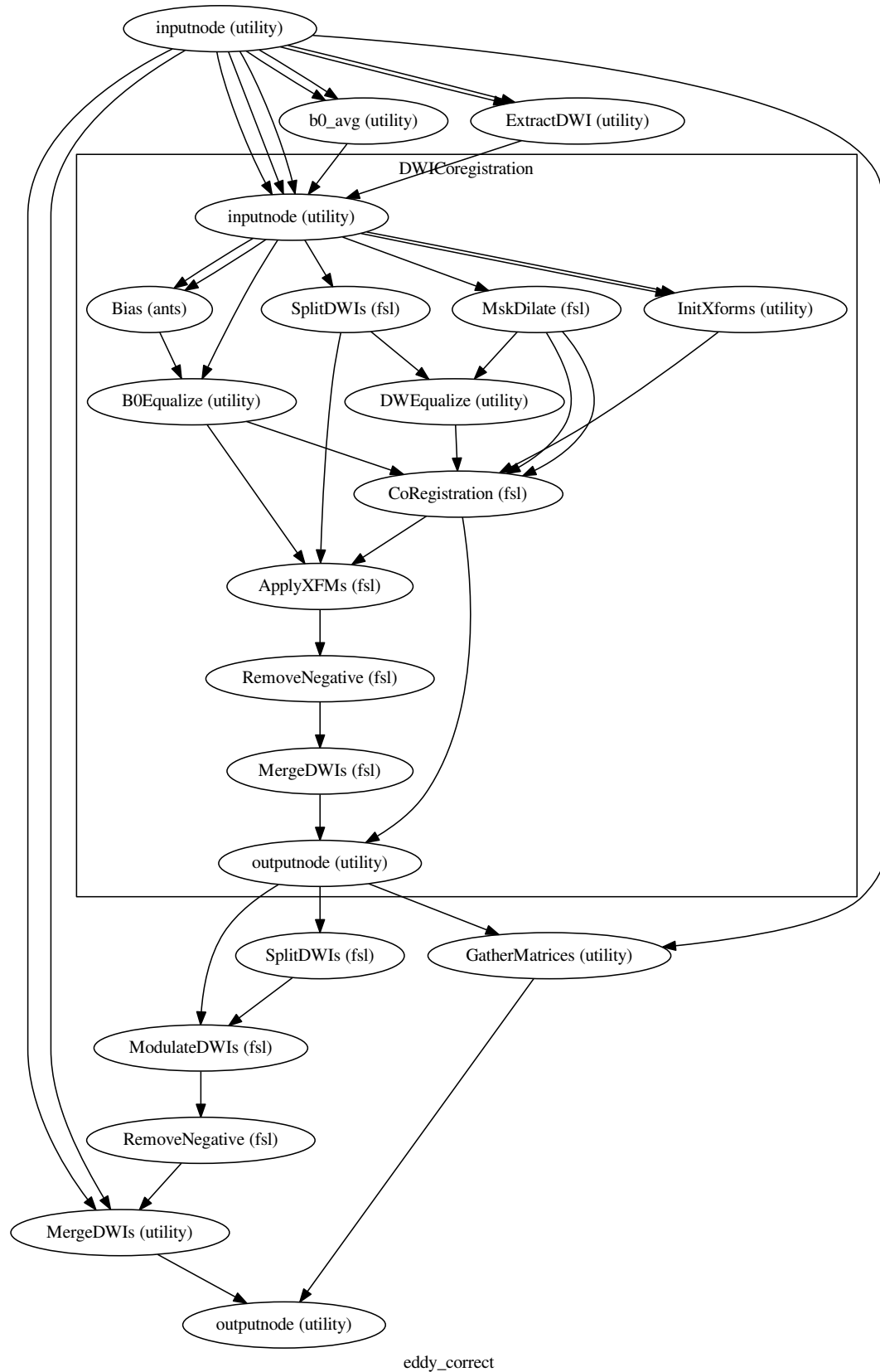
Inputs:

```
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



2.8.5 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()
```

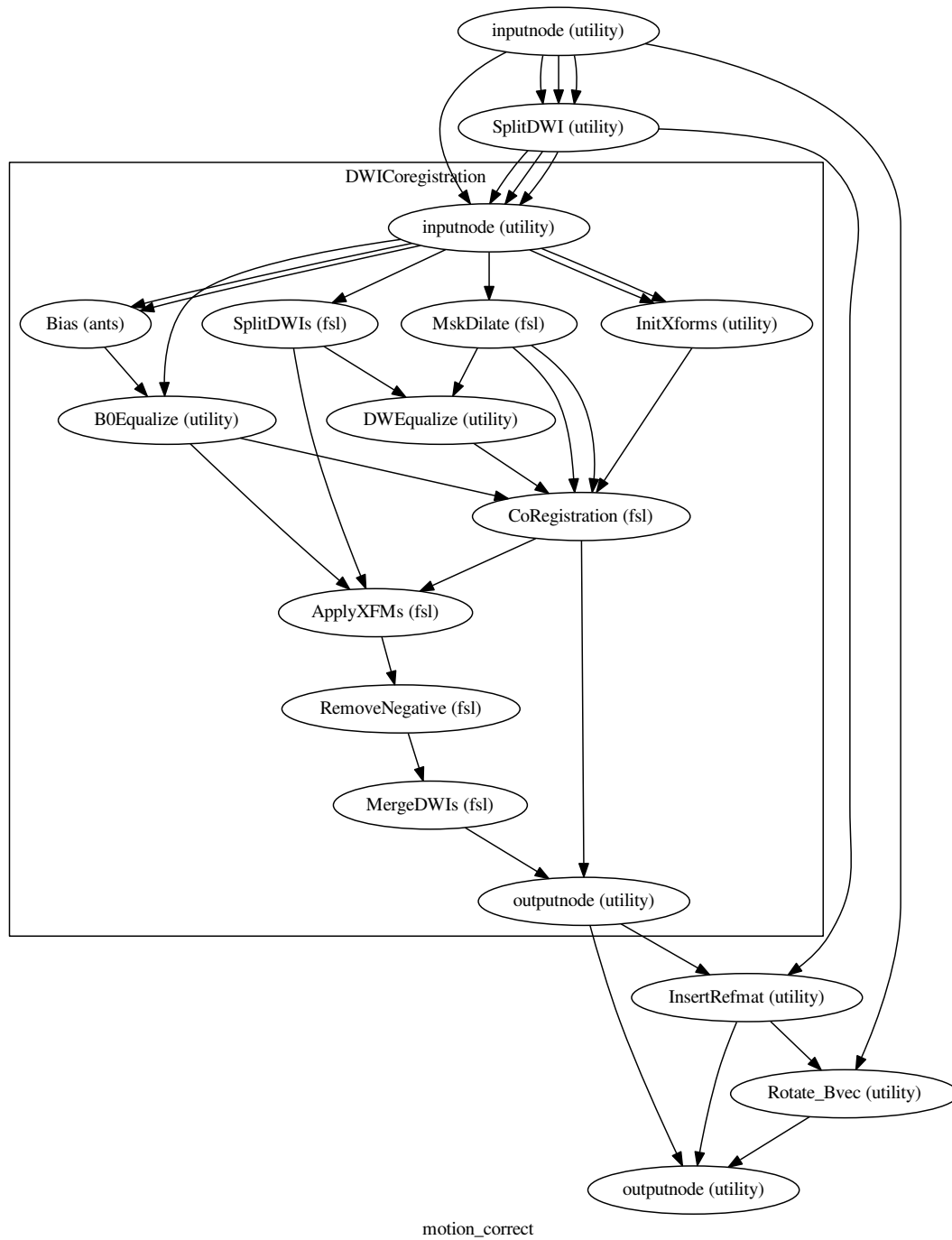
Inputs:

```
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

2.8.6 `remove_bias()`

[Link to code](#)

This workflow estimates a single multiplicative bias field from the averaged *b0* image, as suggested in

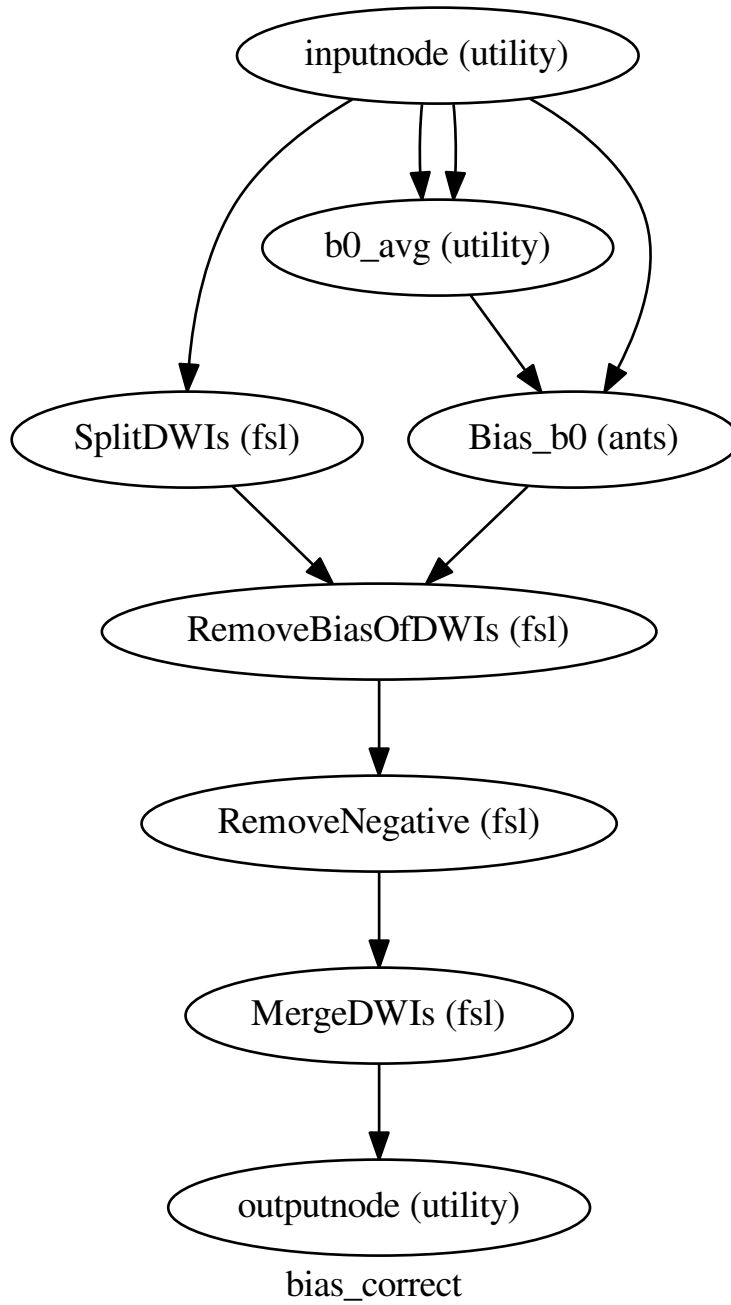
[Jeurissen2014].

References

Example

```
>>> from nipype.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

**2.8.7 `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`.

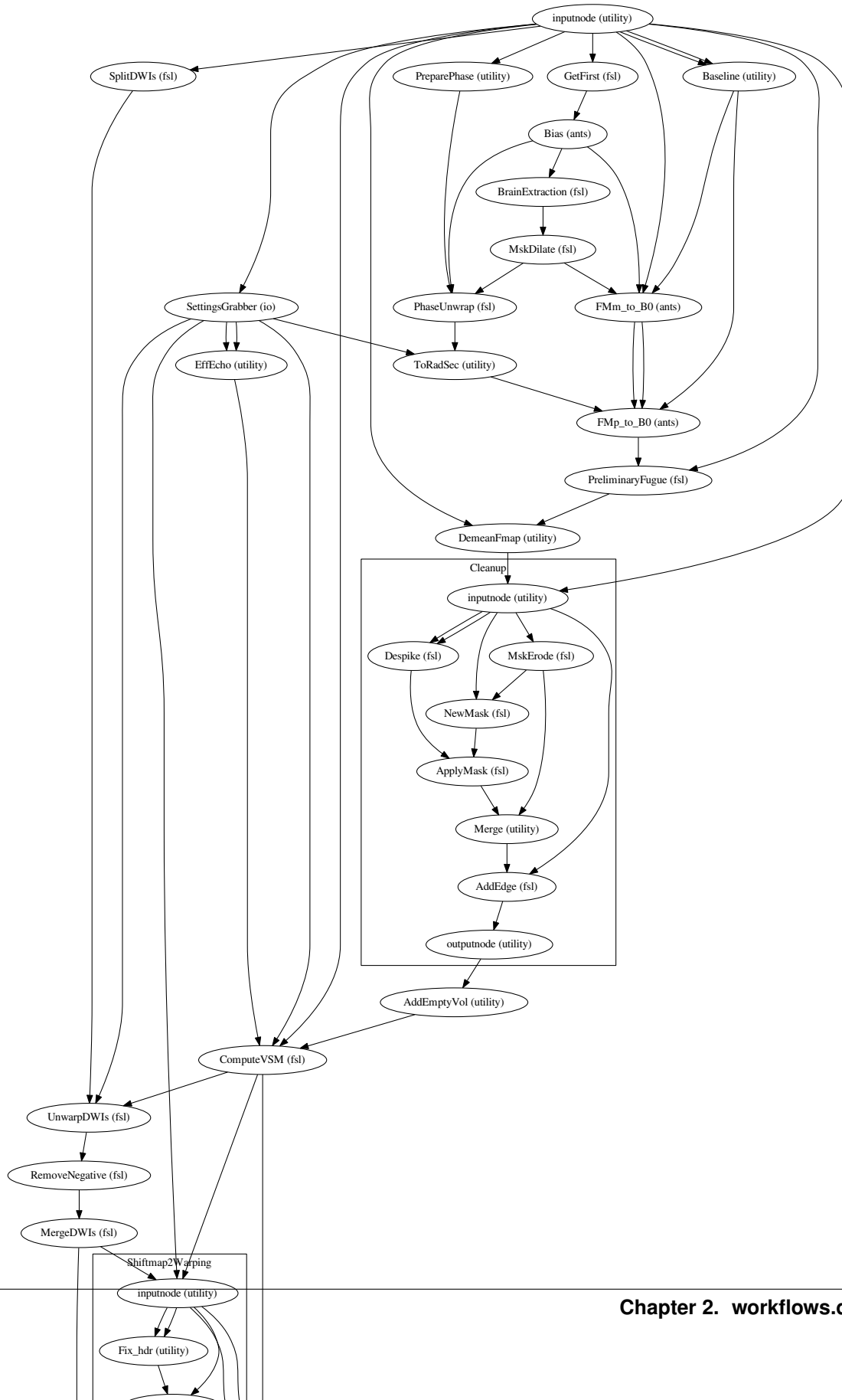
Example

```
>>> from nipype.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



2.8.8 sdc_peb()

[Link to code](#)

SDC stands for susceptibility distortion correction. PEB stands for phase-encoding-based.

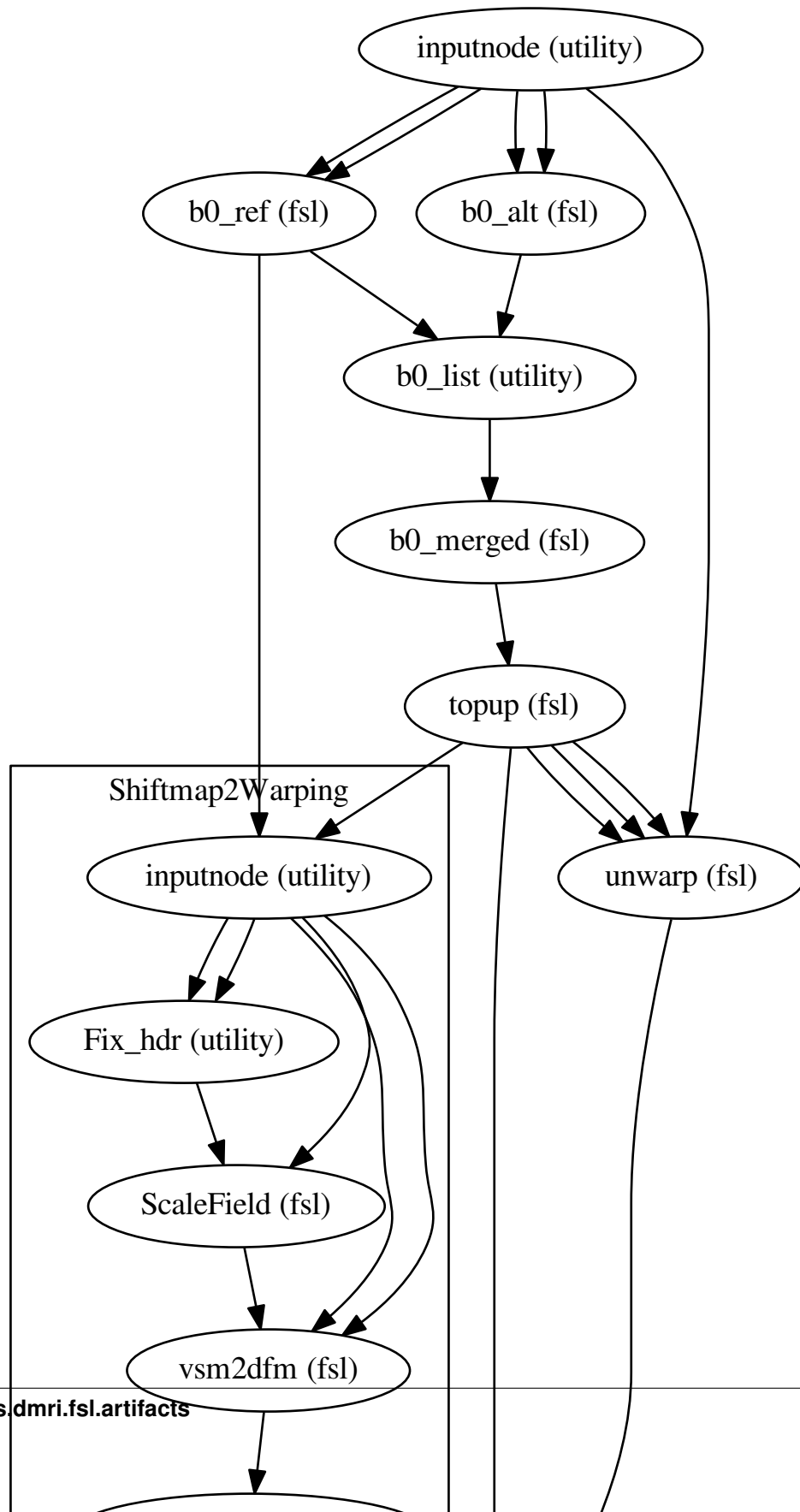
The phase-encoding-based (PEB) method implements SDC by acquiring diffusion images with two different encoding directions [Andersson2003]. The most typical case is acquiring with opposed phase-gradient blips (e.g. $A \ggg P$ and $P \ggg A$, or equivalently, $-y$ and y) as in [Chiou2000], but it is also 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 nipyne.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



2.9 workflows.dmri.fsl.dti

2.9.1 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 nipyne.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('nipyne_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')
```

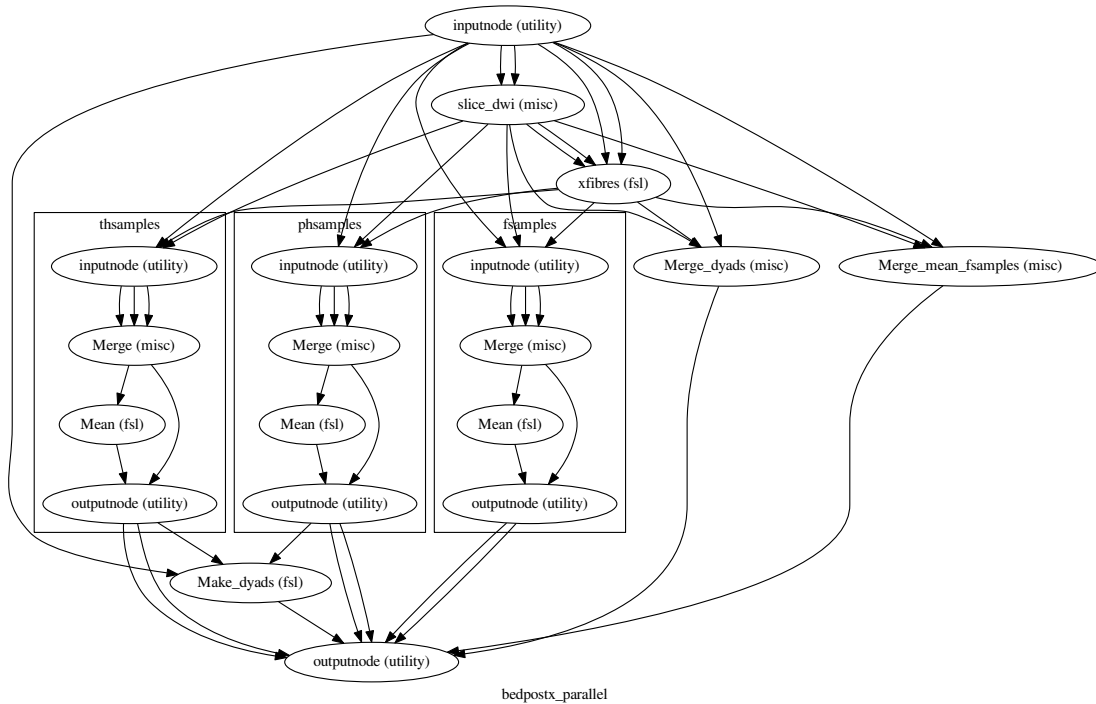
Inputs:

```
inputnode.dwi
inputnode.mask
inputnode.bvecs
inputnode.bvals
```

Outputs:

```
outputnode wraps all XFibres outputs
```

Graph



2.9.2 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:

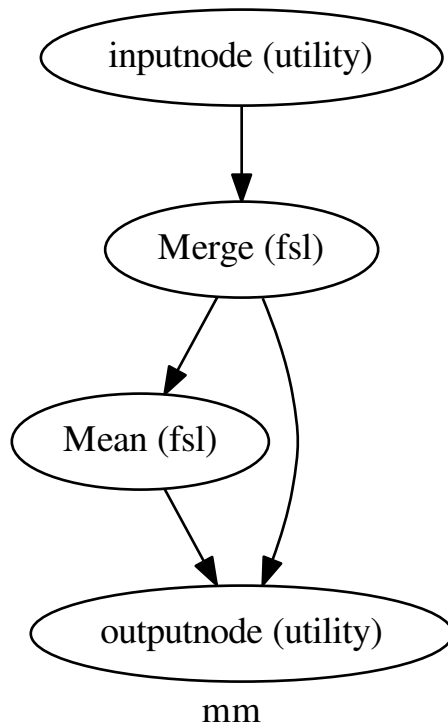
Chapter 2. workflows.dmri

2.9.3 merge_and_mean()

[Link to code](#)



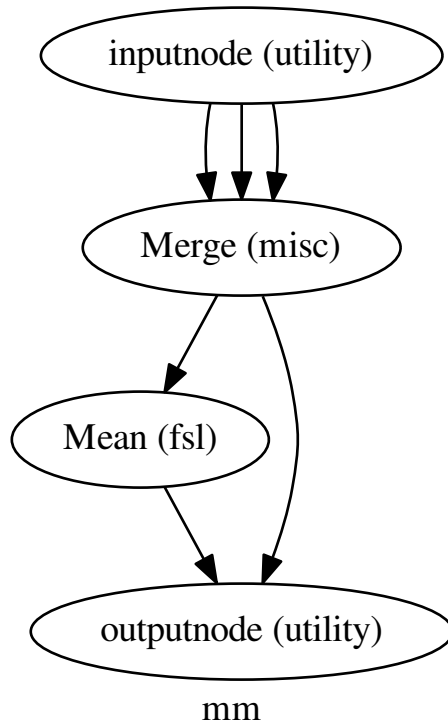
Graph



2.9.4 merge_and_mean_parallel()

[Link to code](#)

Graph



2.9.5 `transpose()`

[Link to code](#)

2.10 `workflows.dmri.fsl.epi`

2.10.1 `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 `nipy.workflow.s.dmri.preprocess.epi.all_fmb_pipeline()` or `nipy.workflow.s.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

```

>>> nipyype_dmri_preprocess = create_dmri_preprocessing('nipyype_dmri_prep')
>>> nipyype_dmri_preprocess.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipyype_dmri_preprocess.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> nipyype_dmri_preprocess.inputs.inputnode.ref_num = 0
>>> nipyype_dmri_preprocess.inputs.inputnode.fieldmap_mag = 'magnitude.nii'
>>> nipyype_dmri_preprocess.inputs.inputnode.fieldmap pha = 'phase.nii'
>>> nipyype_dmri_preprocess.inputs.inputnode.te_diff = 2.46
>>> nipyype_dmri_preprocess.inputs.inputnode.epi_echospacing = 0.77
>>> nipyype_dmri_preprocess.inputs.inputnode.epi_rev_encoding = False
>>> nipyype_dmri_preprocess.inputs.inputnode.pi_accel_factor = True
>>> nipyype_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

```

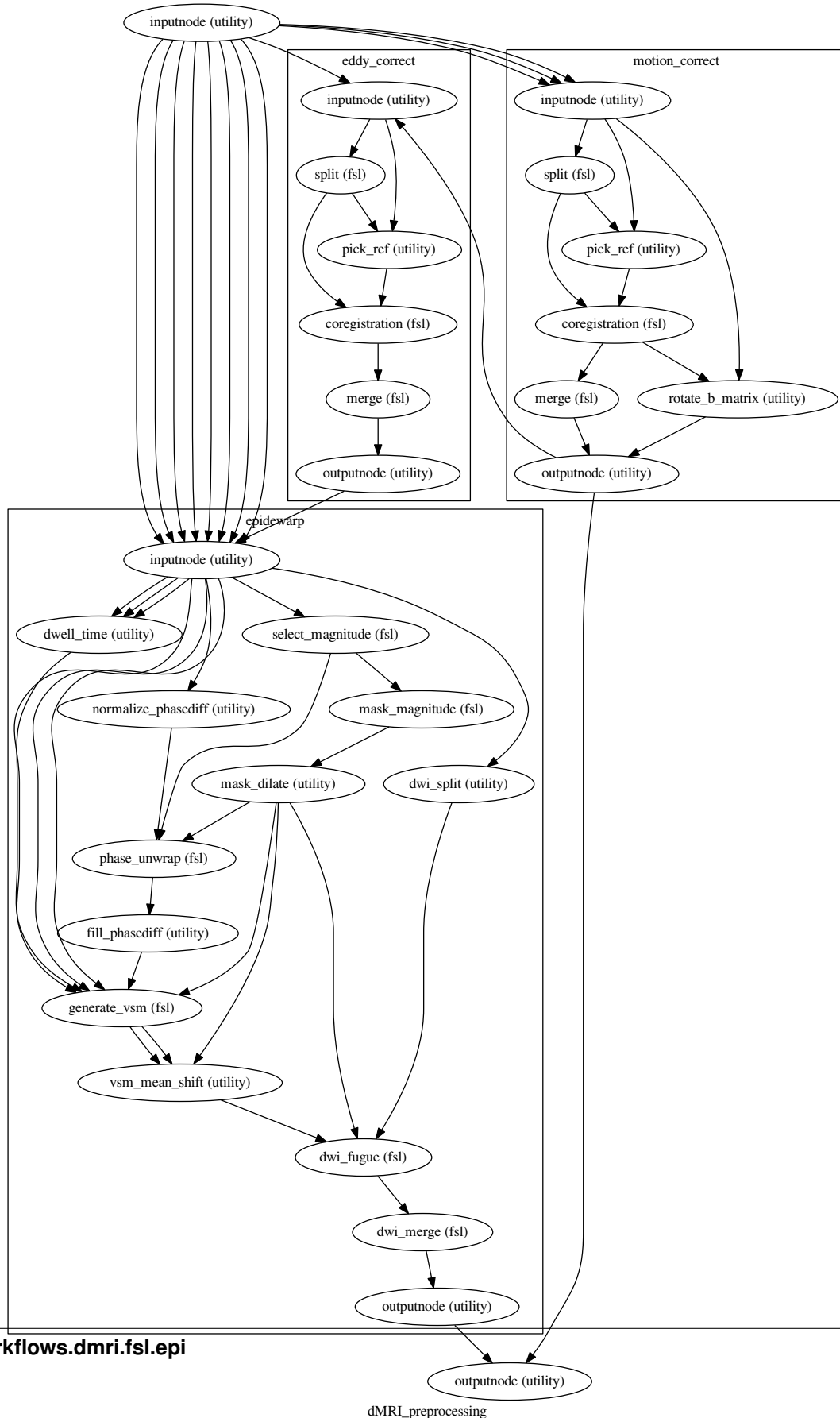
Optional arguments:

```

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



2.10.2 `create_eddy_correct_pipeline()`

[Link to code](#)

Deprecated since version 0.9.3: Use `nipy.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

```
>>> nipy_ecc = create_eddy_correct_pipeline('nipy_ecc')
>>> nipy_ecc.inputs.in_file = 'diffusion.nii'
>>> nipy_ecc.inputs.ref_num = 0
>>> nipy_ecc.run()
```

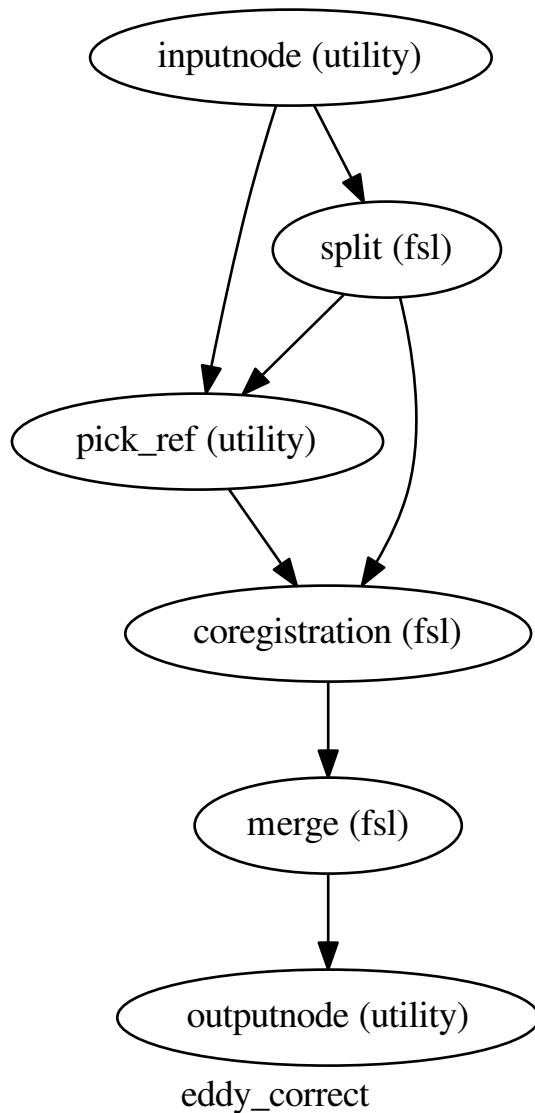
Inputs:

```
inputnode.in_file
inputnode.ref_num
```

Outputs:

```
outputnode.eddy_corrected
```

Graph



2.10.3 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)

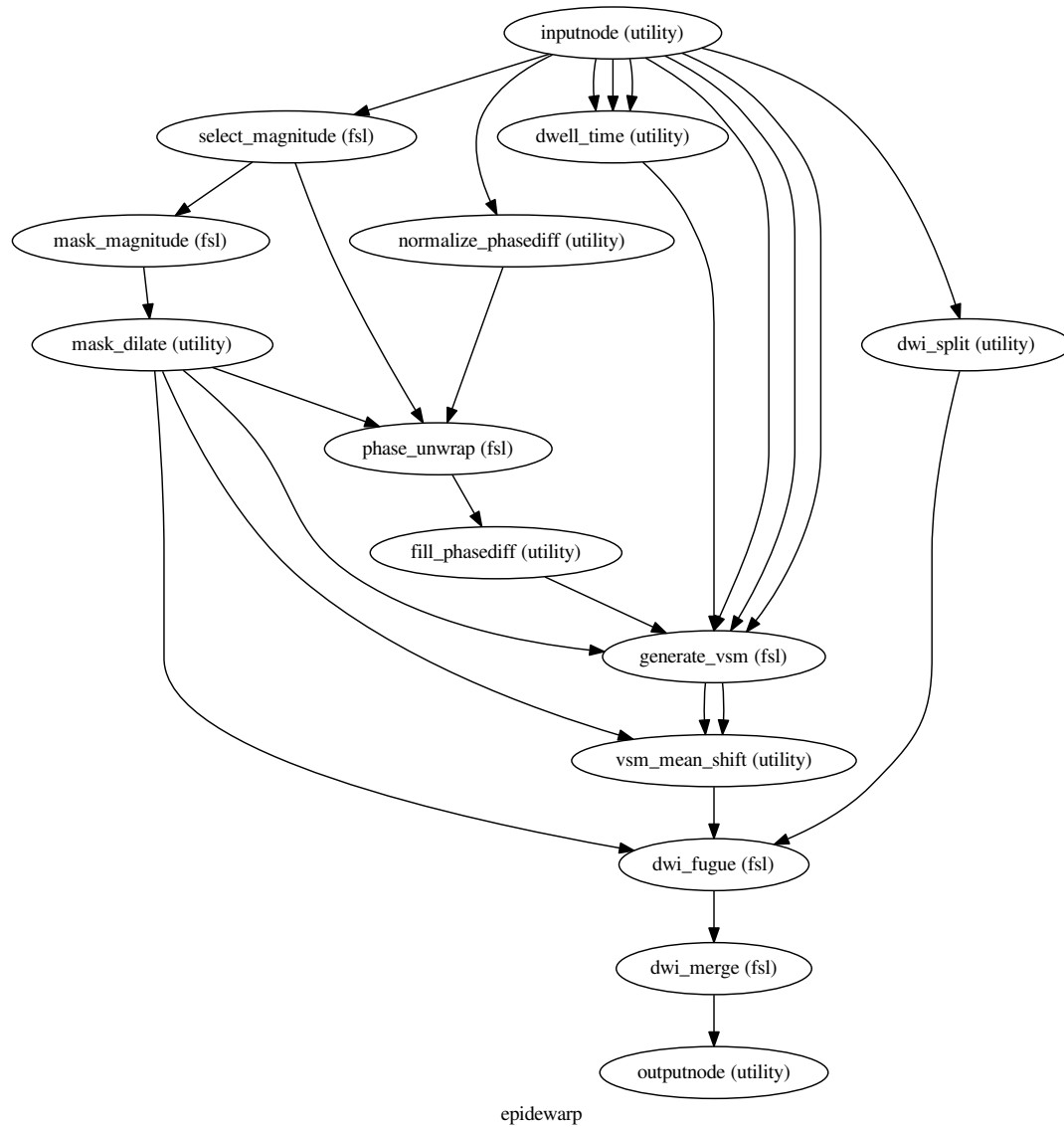
Outputs:

`outputnode.epi_corrected`

Optional arguments:

`fieldmap_registration` - **True** if registration to fieldmap should be done (default,
↳ **False**)

Graph



2.10.4 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

```
>>> nipy_motioncorrect = create_motion_correct_pipeline('nipy_motioncorrect')
>>> nipy_motioncorrect.inputs.inputnode.in_file = 'diffusion.nii'
>>> nipy_motioncorrect.inputs.inputnode.in_bvec = 'diffusion.bvec'
>>> nipy_motioncorrect.inputs.inputnode.ref_num = 0
>>> nipy_motioncorrect.run()
```

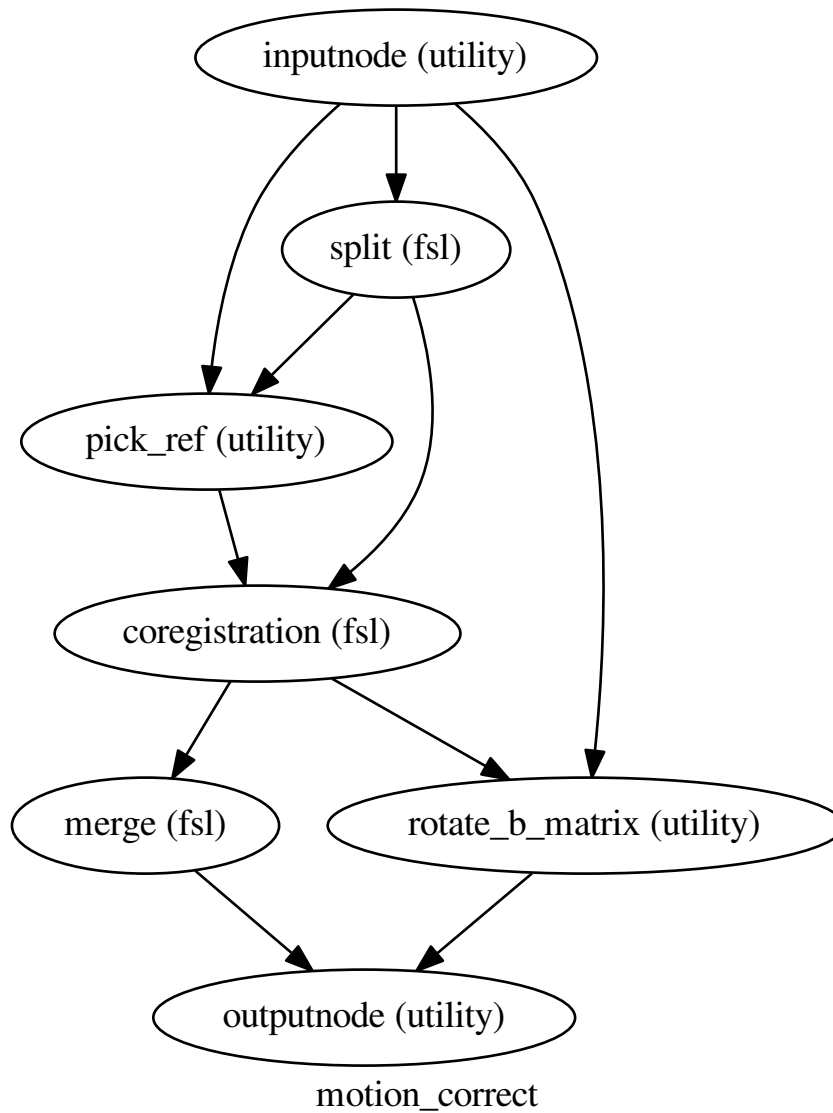
Inputs:

```
inputnode.in_file
inputnode.ref_num
inputnode.in_bvec
```

Outputs:

```
outputnode.motion_corrected
outputnode.out_bvec
```


Graph



2.10.5 fieldmap_correction()

[Link to code](#)

Deprecated since version 0.9.3: Use `nipyype.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()
```

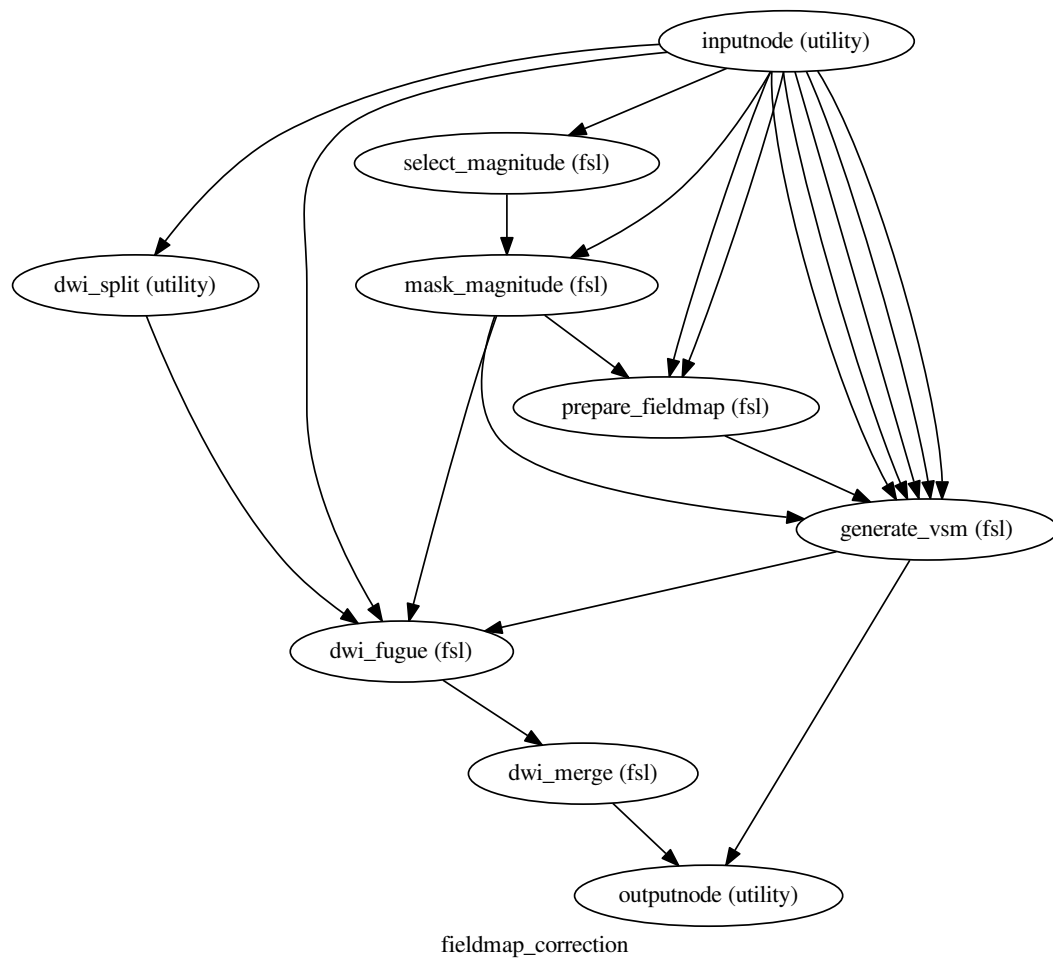
Inputs:

```
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 magnitude 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



2.10.6 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()
  
```

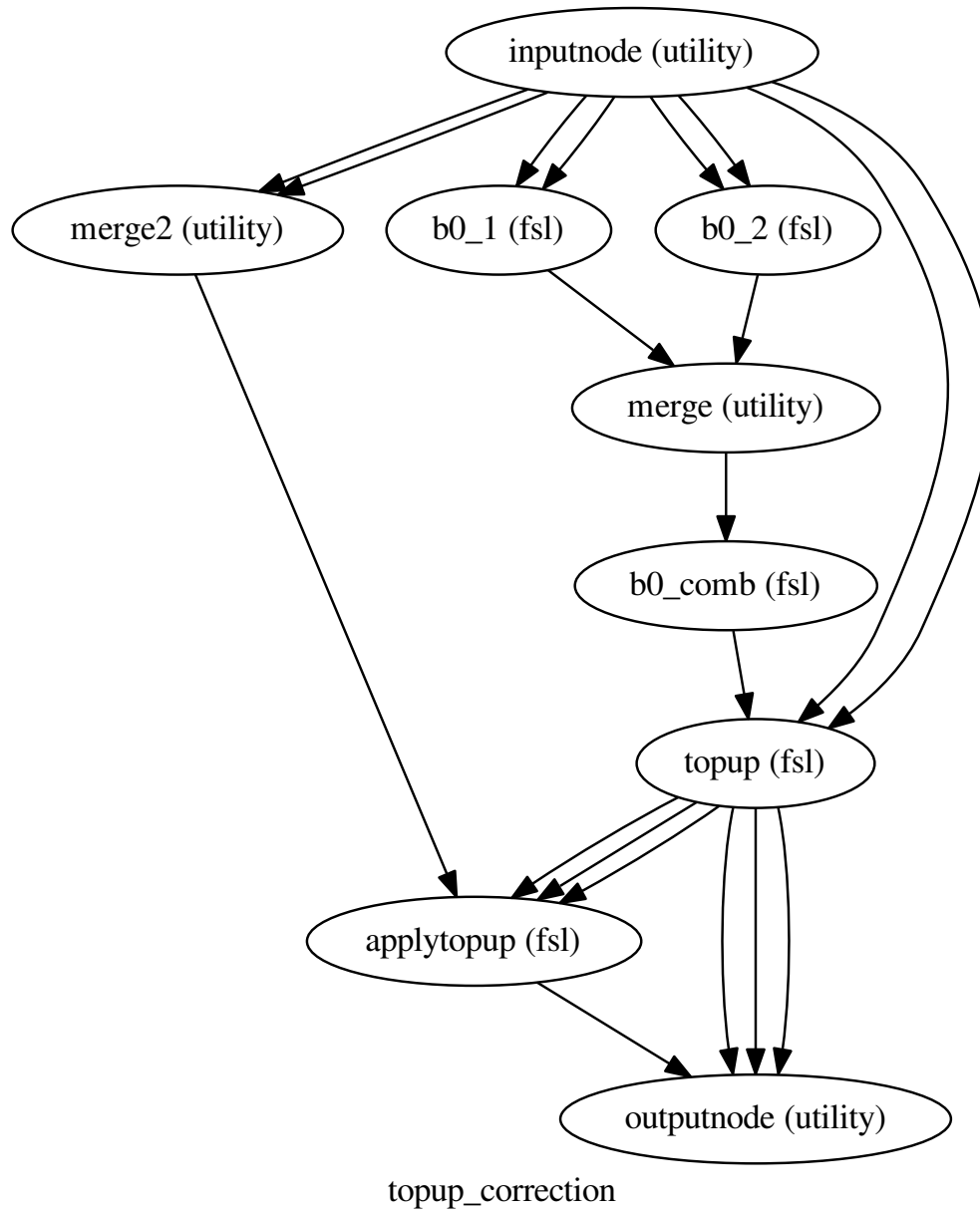
Inputs:

```
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



2.11 workflows.dmri.fsl.tbss

2.11.1 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']
```

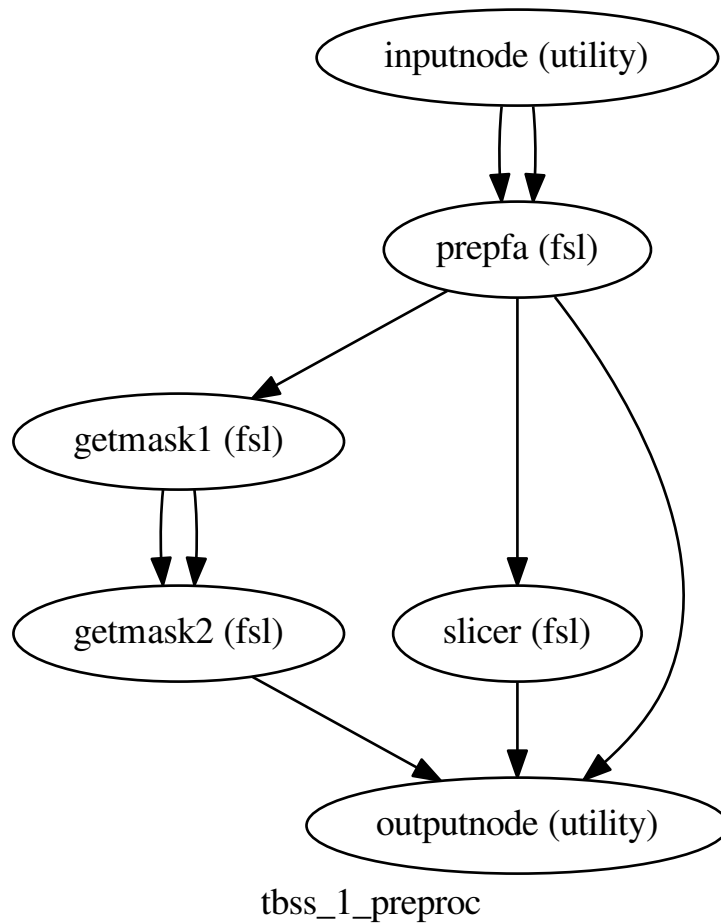
Inputs:

```
inputnode.fa_list
```

Outputs:

```
outputnode.fa_list
outputnode.mask_list
outputnode.slices
```

Graph



2.11.2 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']

```

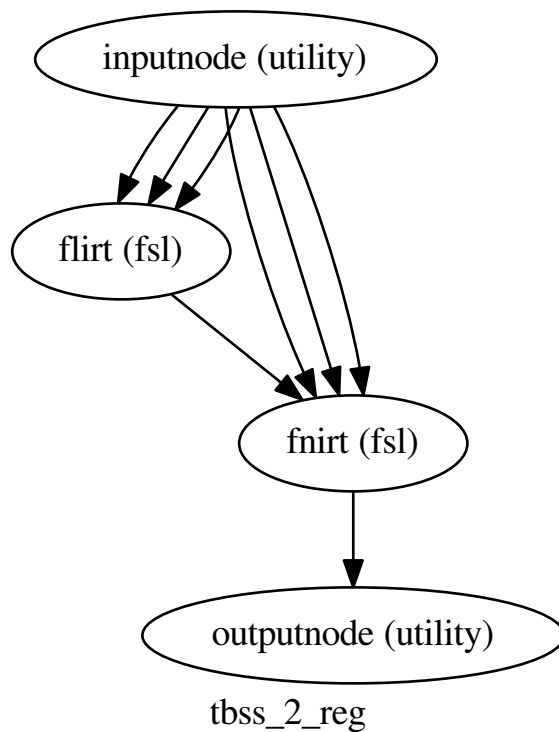
Inputs:

```
inputnode.fa_list  
inputnode.mask_list  
inputnode.target
```

Outputs:

```
outputnode.field_list
```

Graph



2.11.3 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 nipyne.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']
```

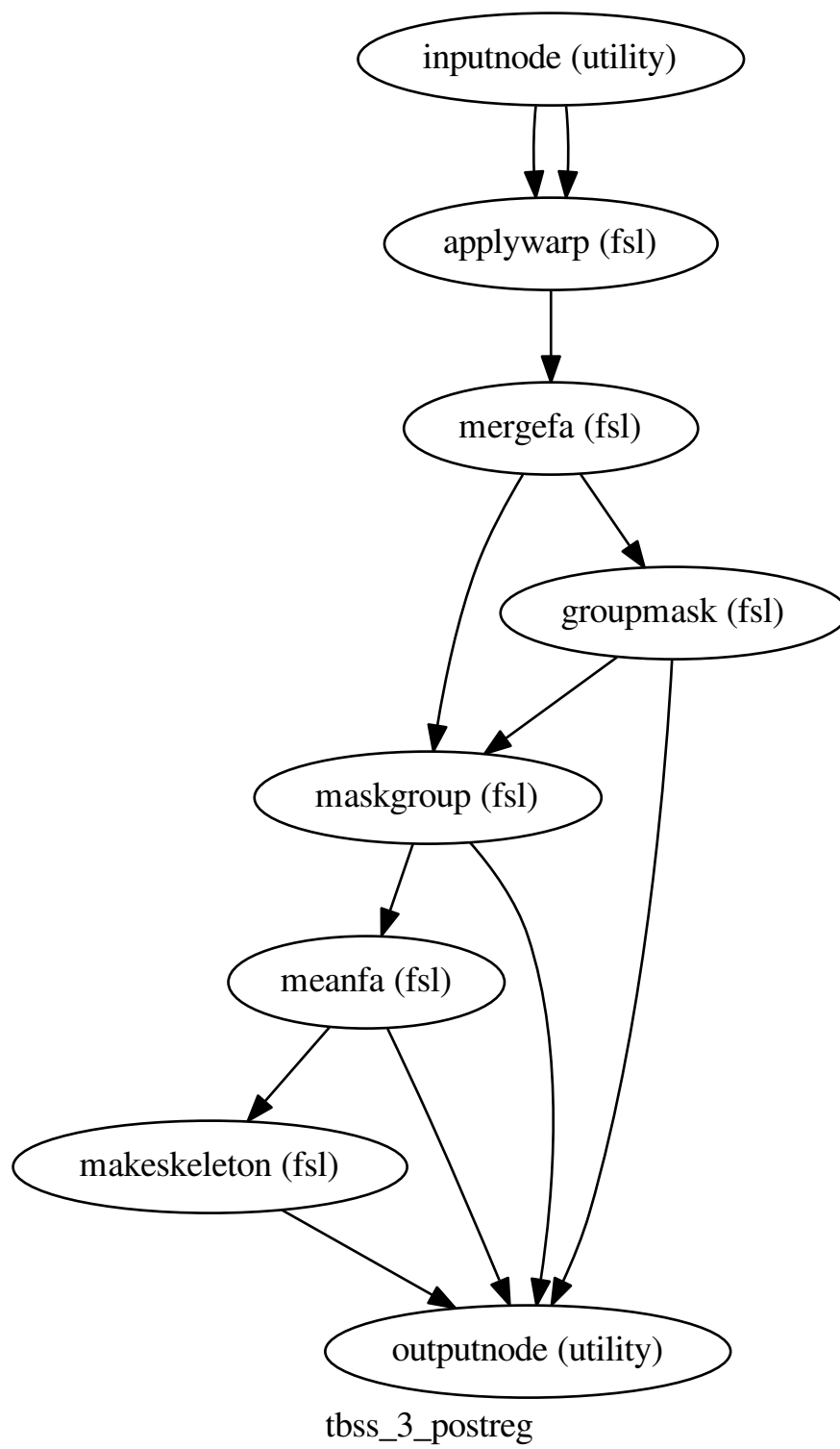
Inputs:

```
inputnode.field_list
inputnode.fa_list
```

Outputs:

```
outputnode.groupmask
outputnode.skeleton_file
outputnode.meanfa_file
outputnode.mergefa_file
```


Graph



2.11.4 `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

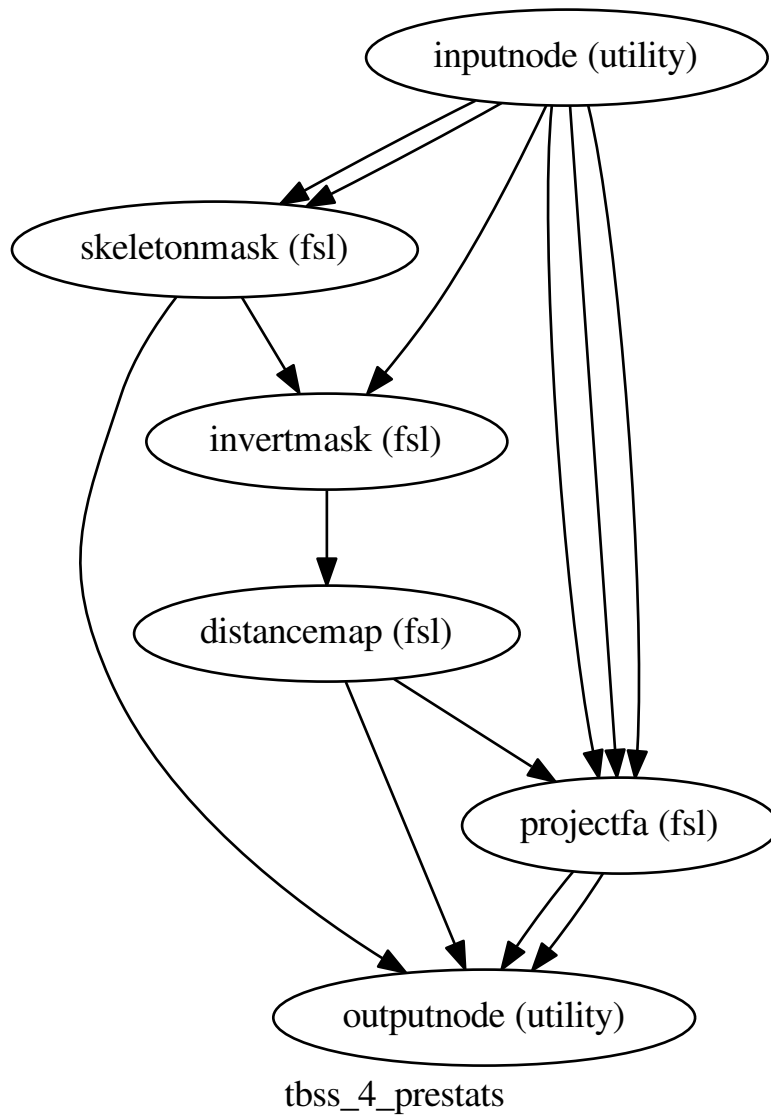
```
>>> from nipy.workflows.dmri.fsl import tbss
>>> tbss4 = tbss.create_tbss_4_prestats(name='tbss4')
>>> tbss4.inputs.inputnode.skeleton_thresh = 0.2
```

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**2.11.5 create_tbss_all()**[Link to code](#)

Create a pipeline that combines create_tbss_* pipelines

Example

```
>>> from nipy.workflows.dmri.fsl import tbss
>>> tbss_wf = tbss.create_tbss_all('tbss', estimate_skeleton=True)
>>> tbss_wf.inputs.inputnode.skeleton_thresh = 0.2
>>> tbss_wf.inputs.inputnode.fa_list = ['s1_wrapped_FA.nii', 's2_wrapped_FA.nii',
↪ 's3_wrapped_FA.nii']
```

```
>>> tbss_wf = tbss.create_tbss_all('tbss', estimate_skeleton=False)
>>> tbss_wf.inputs.inputnode.skeleton_thresh = 0.2
>>> tbss_wf.inputs.inputnode.fa_list = ['s1_wrapped_FA.nii', 's2_wrapped_FA.nii',
↪ 's3_wrapped_FA.nii']
```

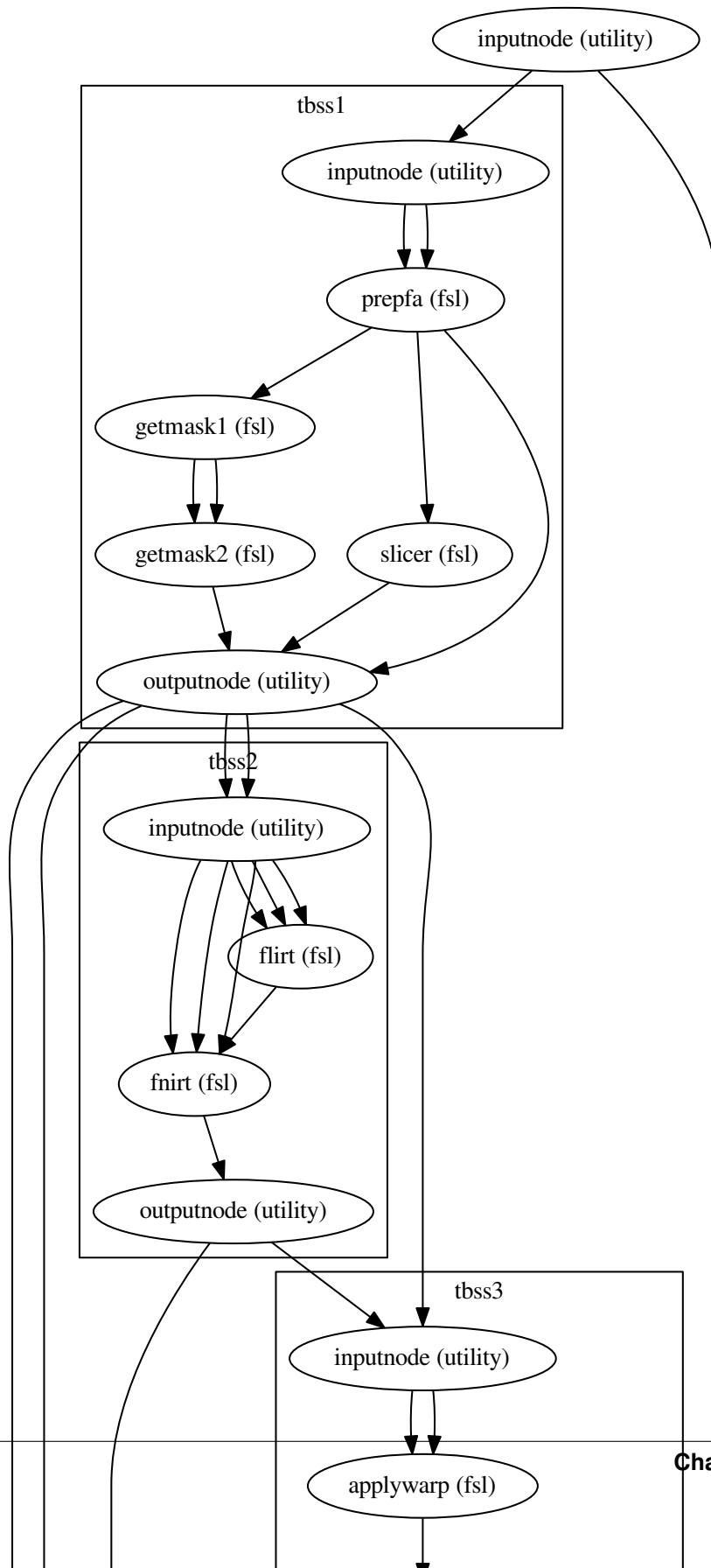
Inputs:

```
inputnode.fa_list
inputnode.skeleton_thresh
```

Outputs:

```
outputnode.meanfa_file
outputnode.projectedfa_file
outputnode.skeleton_file
outputnode.skeleton_mask
```


Graph



2.11.6 create_tbss_non_FA()

[Link to code](#)

A pipeline that implement tbss_non_FA in FSL

Example

```
>>> from nipyre.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'
```

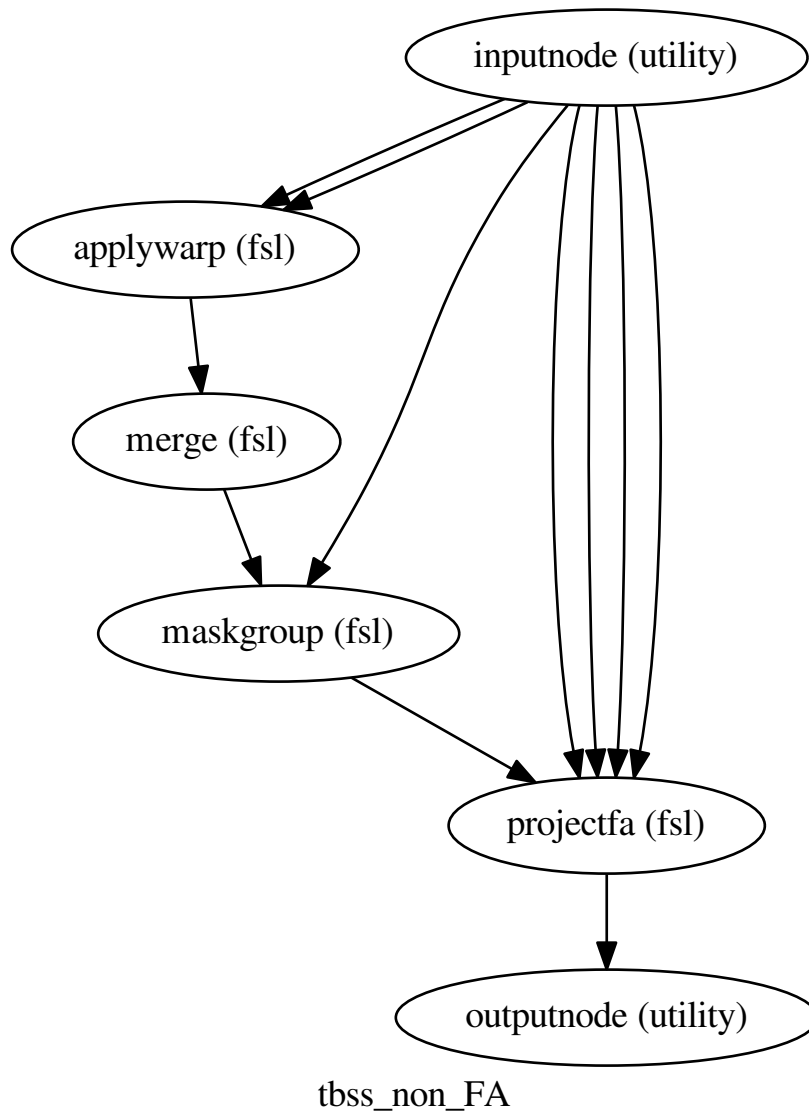
Inputs:

```
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



2.11.7 tbss1_op_string()

[Link to code](#)

2.11.8 tbss4_op_string()

[Link to code](#)

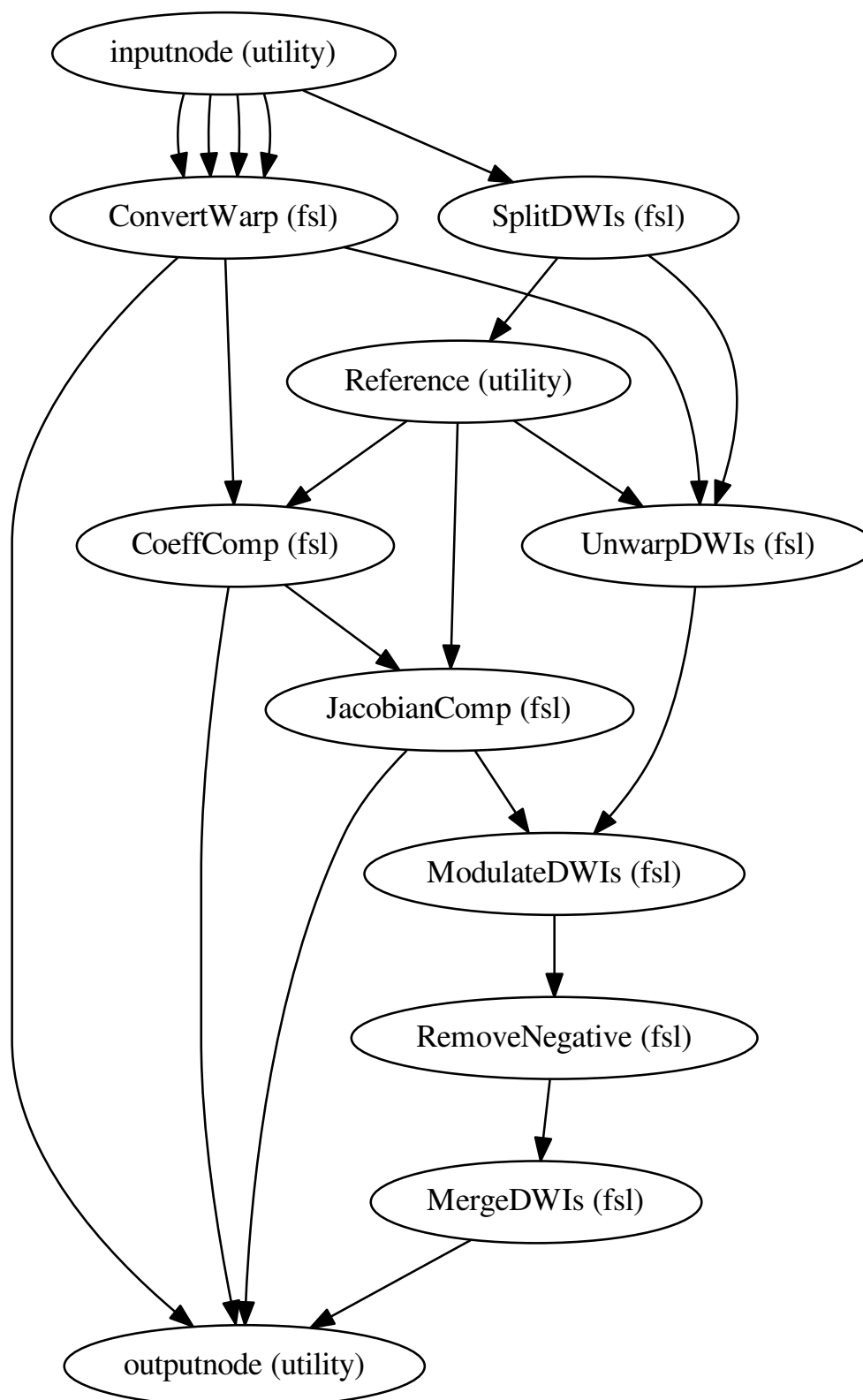
2.12 workflows.dmri.fsl.utils

2.12.1 `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



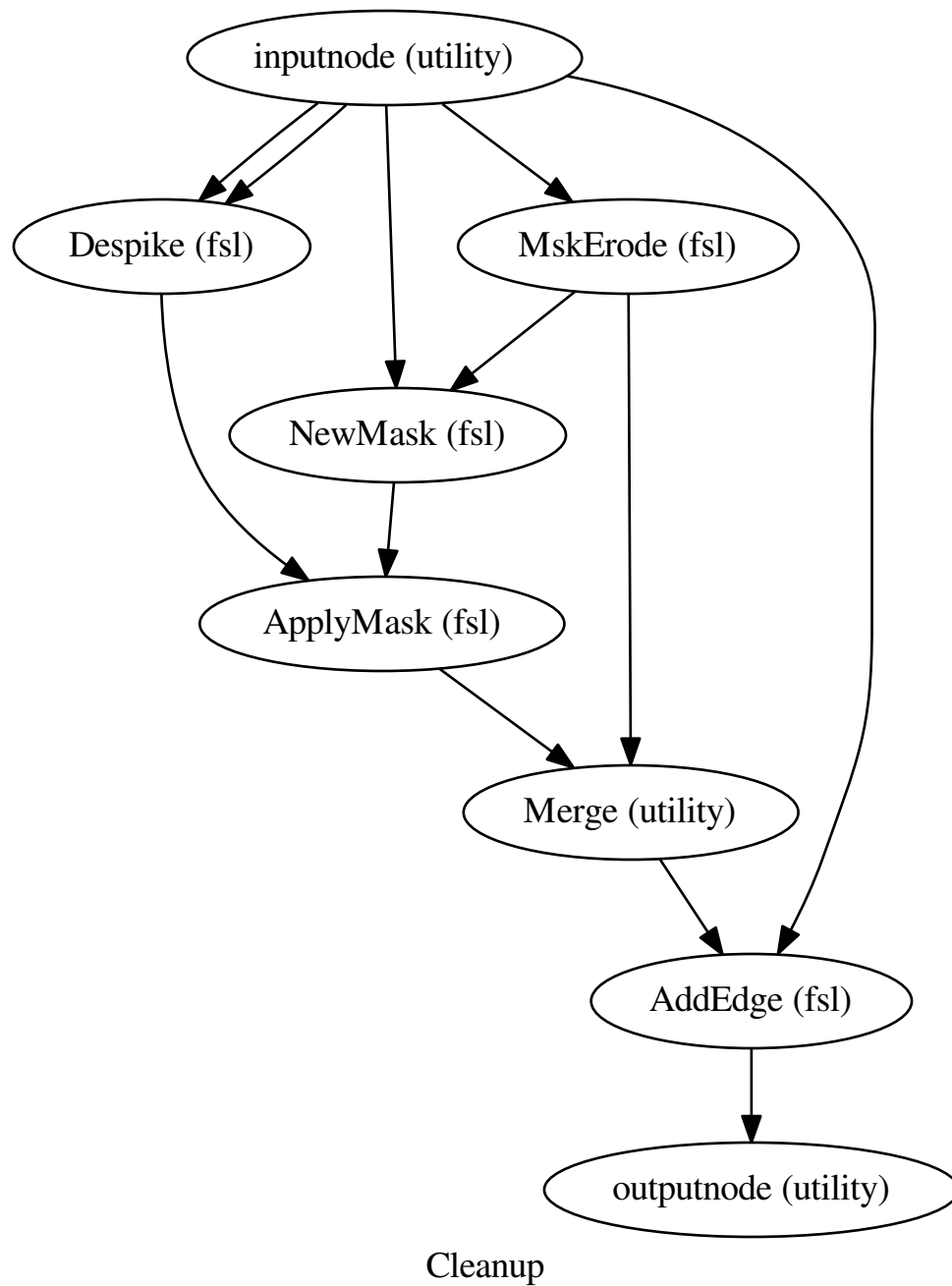
UnwarpArtifacts

2.12.2 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

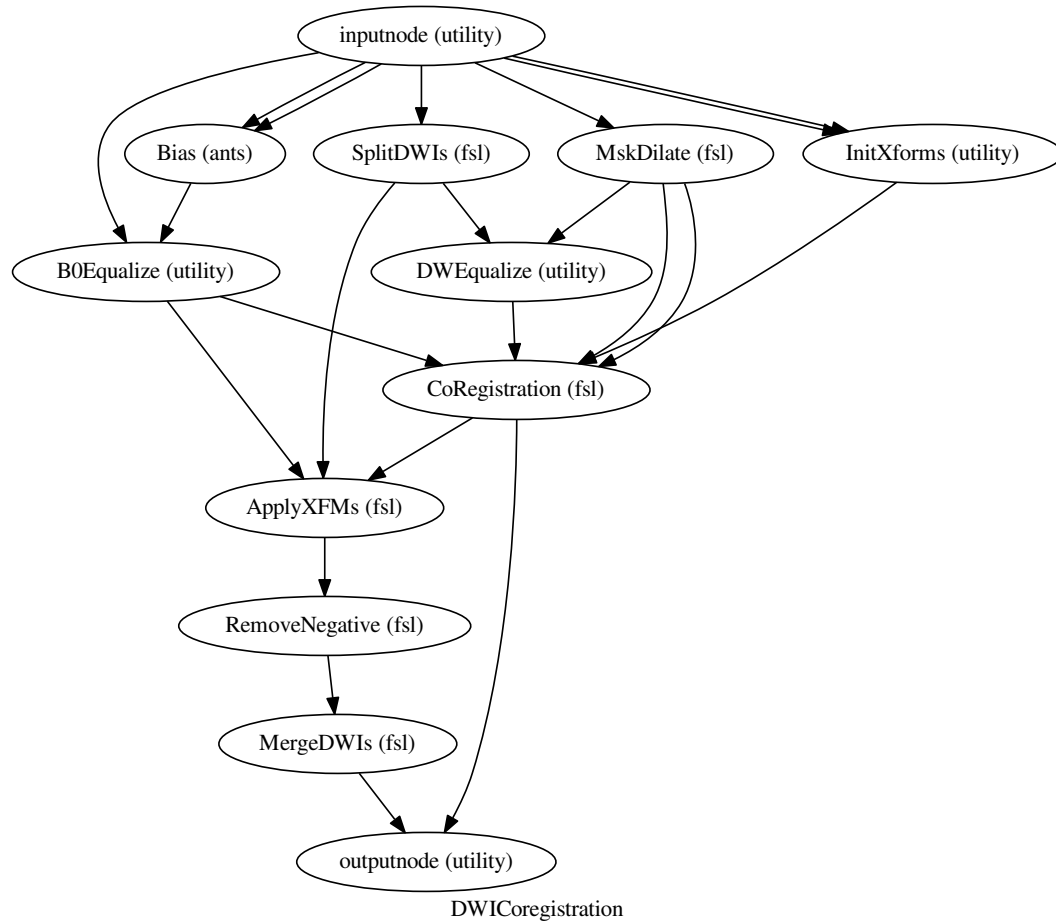


2.12.3 dwi_flirt()

[Link to code](#)

Generates a workflow for linear registration of dwi volumes

Graph

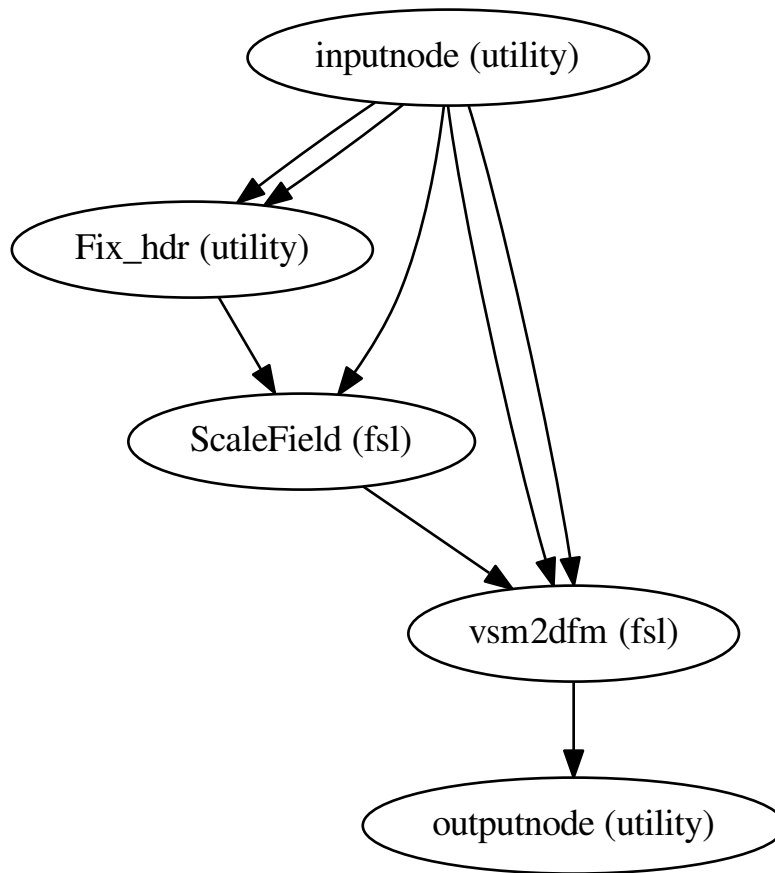


2.12.4 vsm2warp()

[Link to code](#)

Converts a voxel shift map (vsm) to a displacements field (warp).

Graph



Shiftmap2Warping

2.12.5 `add_empty_vol()`

[Link to code](#)

Adds an empty vol to the phase difference image

2.12.6 `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: <i>b0</i> should be already registered (head motion artifact should be corrected).
--

2.12.7 `b0_indices()`

[Link to code](#)

Extract the indices of slices in a b-values file with a low b value

2.12.8 `compute_readout()`

[Link to code](#)

Computes readout time from epi params (see [eddy documentation](#)).

Warning: `params['echospadding']` should be in *sec* units.

2.12.9 `copy_hdr()`

[Link to code](#)

2.12.10 `demean_image()`

[Link to code](#)

Demean image data inside mask

2.12.11 `eddy_rotate_bvecs()`

[Link to code](#)

Rotates the input bvec file accordingly with a list of parameters sourced from `eddy`, as explained [here](#).

2.12.12 `enhance()`

[Link to code](#)

2.12.13 `extract_bval()`

[Link to code](#)

Writes an image containing only the volumes with b-value specified at input

2.12.14 `hmc_split()`

[Link to code](#)

Selects the reference and moving volumes from a dwi dataset for the purpose of HMC.

2.12.15 `insert_mat()`

[Link to code](#)

2.12.16 `rads2radsec()`

[Link to code](#)

Converts input phase difference map to rads

2.12.17 `recompose_dwi()`

[Link to code](#)

Recompose back the dMRI data accordingly the b-values table after EC correction

2.12.18 `recompose_xfm()`

[Link to code](#)

Insert identity transformation matrices in b0 volumes to build up a list

2.12.19 `remove_comp()`

[Link to code](#)

Removes the volume `valid` from the 4D nifti file

2.12.20 `reorient_bvecs()`

[Link to code](#)

Checks reorientations of `in_dwi` w.r.t. `old_dwi` and reorients the `in_bvec` table accordingly.

2.12.21 `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} .

2.12.22 `siemens2rads()`

[Link to code](#)

Converts input phase difference map to rads

2.12.23 `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).

2.13 `workflows.dmri.mrtrix.connectivity_mapping`

2.13.1 `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
>>> connmapper = create_connectivity_pipeline("nipy_conmap")
>>> connmapper.inputs.inputnode.subjects_dir = '.'
>>> connmapper.inputs.inputnode.subject_id = 'subj1'
```

(continues on next page)

(continued from previous page)

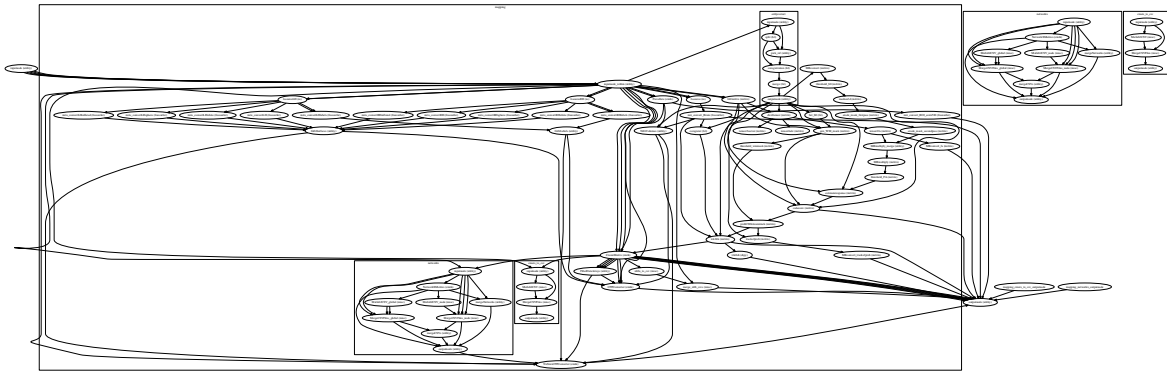
```
>>> conmapper.inputs.inputnode.dwi = 'data.nii.gz'
>>> conmapper.inputs.inputnode.bvecs = 'bvecs'
>>> conmapper.inputs.inputnode.bvals = 'bvals'
>>> conmapper.run()
```

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

2.14 workflows.dmri.mrtrix.diffusion

2.14.1 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()
```

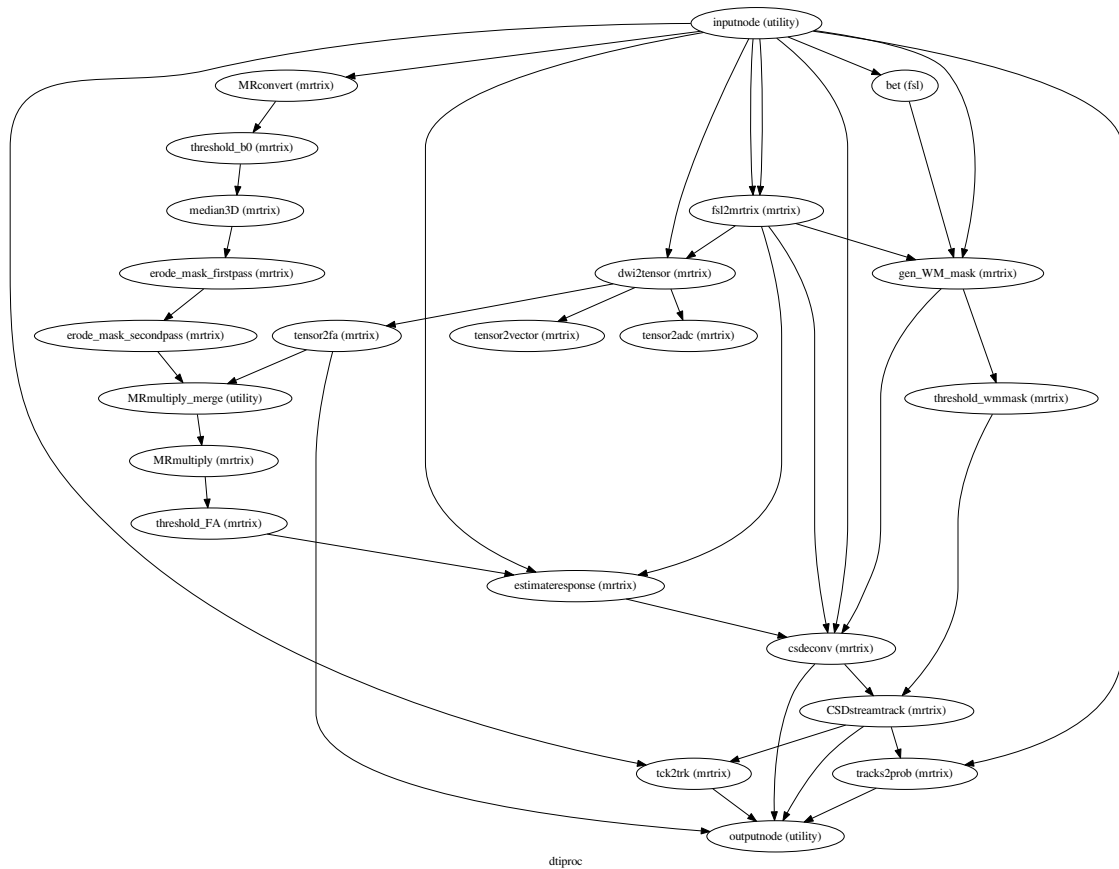
Inputs:

```
inputnode.dwi
inputnode.bvecs
inputnode.bvals
```

Outputs:

```
outputnode.fa
outputnode.tdi
outputnode.tracks_tck
outputnode.tracks_trk
outputnode.csdeconv
```

Graph



2.15 workflows.dmri.mrtrix.group_connectivity

2.15.1 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 nipype.interfaces.freesurfer as fs
>>> import nipype.workflows.dmri.mrtrix.group_connectivity as groupwork
>>> import cmp
>>> from nipype.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']])
```


3.1 workflows.fmri.fsl.estimate

3.1.1 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()
```

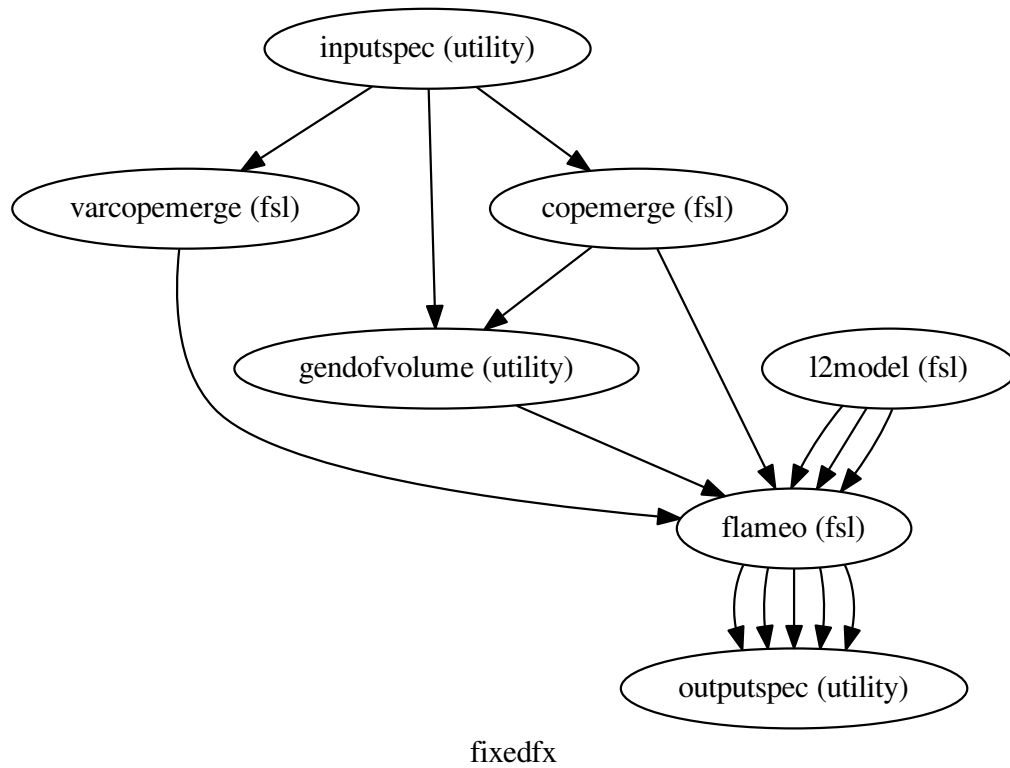
Inputs:

```
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



3.1.2 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

```

(continues on next page)

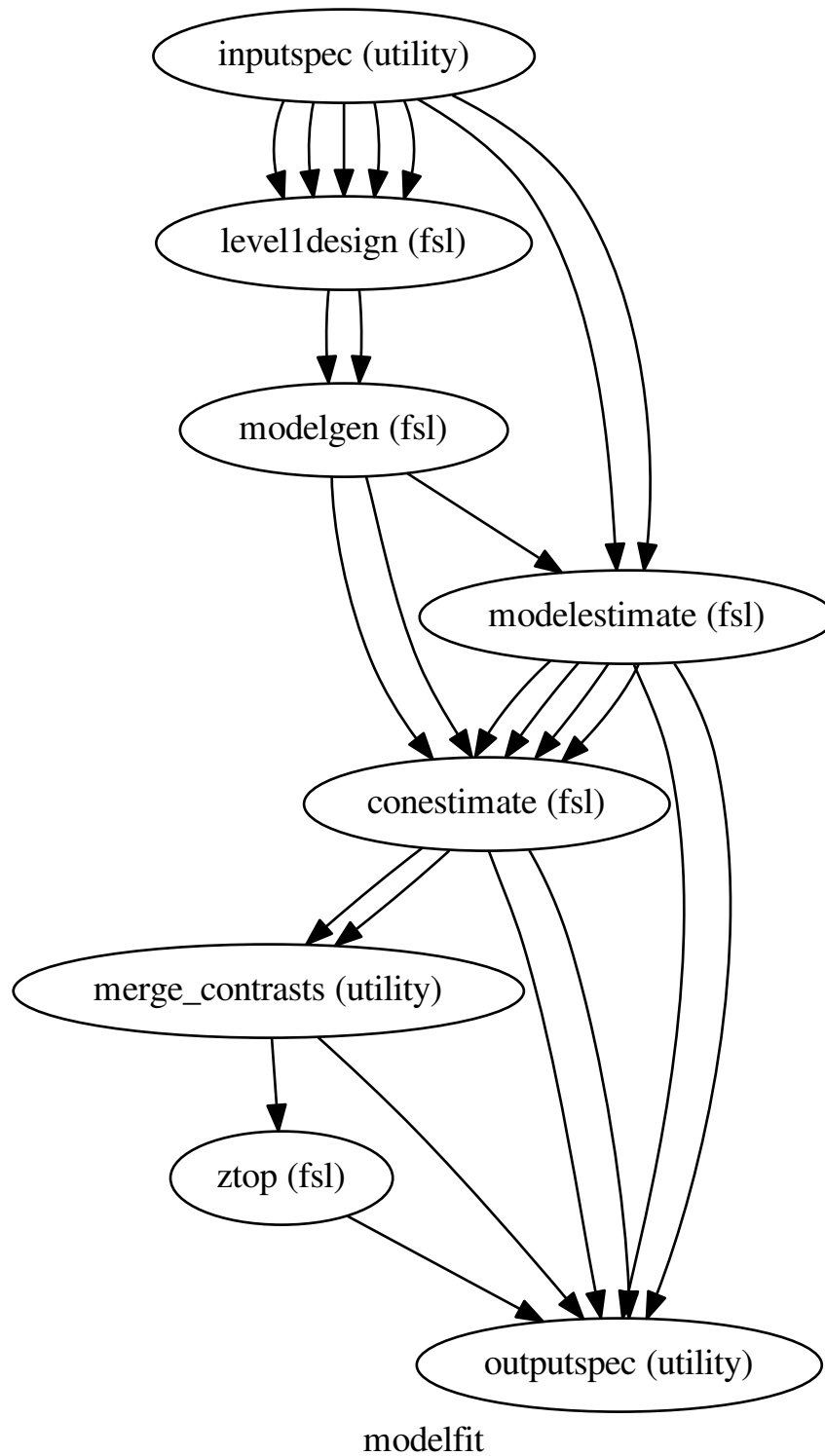
(continued from previous page)

```
inputspec.model_serial_correlations  
inputspec.bases
```

Outputs:

```
outputspec.copes  
outputspec.varcopes  
outputspec.dof_file  
outputspec.pfiles  
outputspec.zfiles  
outputspec.parameter_estimates
```


Graph

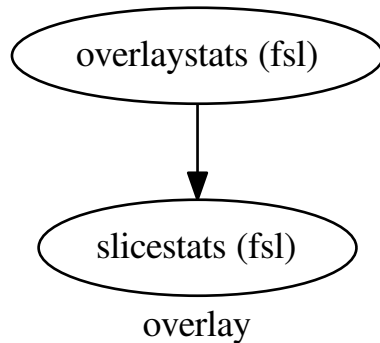


3.1.3 create_overlay_workflow()

[Link to code](#)

Setup overlay workflow

Graph



3.2 workflows.fmri.fsl.preprocess

3.2.1 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')
whichrun : which run to draw reference volume from (integer index or 'first',
↔ 'middle', 'last')
```

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
```

(continues on next page)

(continued from previous page)

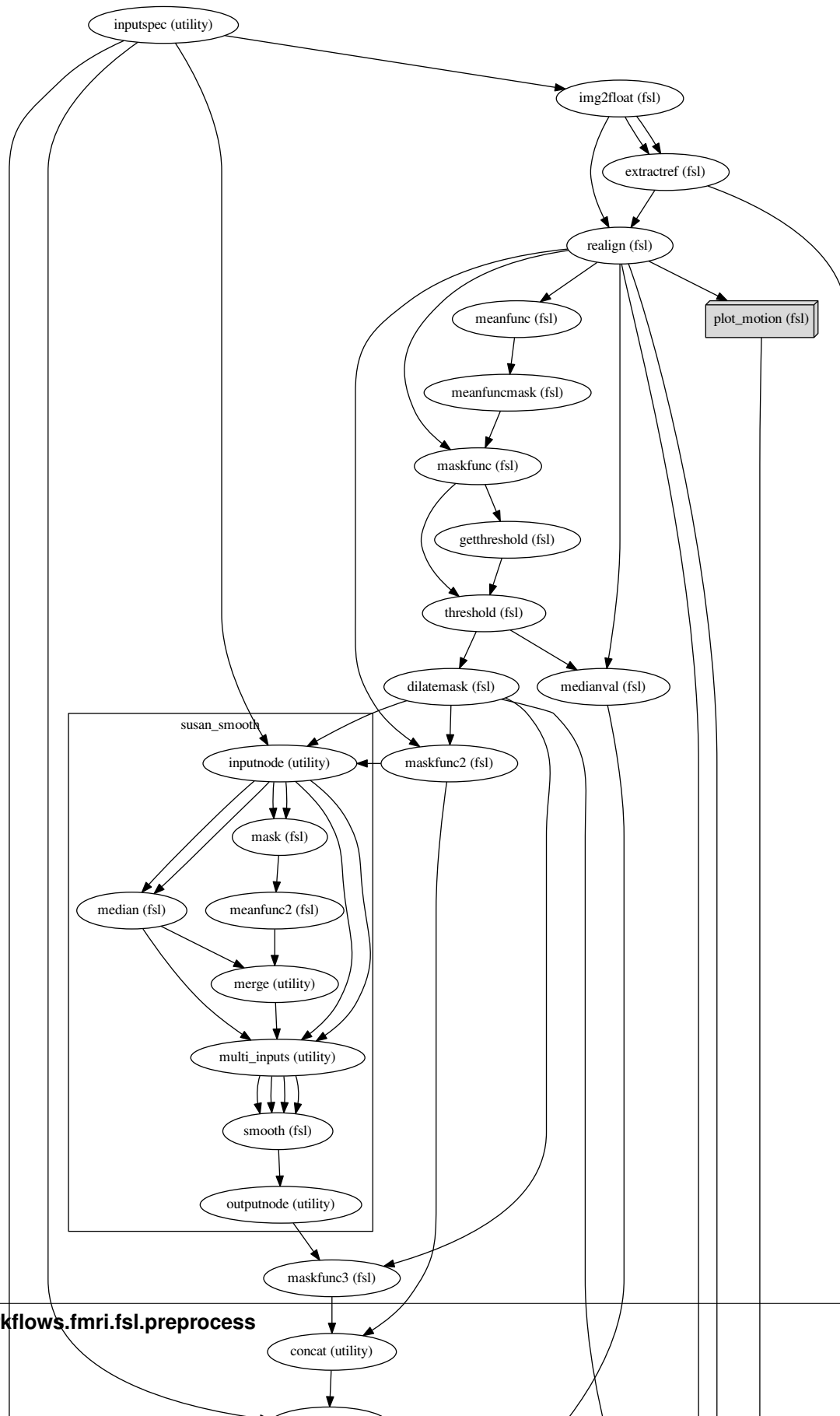
```
outputspec.highpassed_files : highpassed functional data (if highpass=True)
outputspec.mean : mean file
```

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()
```


Graph



3.2.2 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



3.2.3 create_parallelfcat_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:

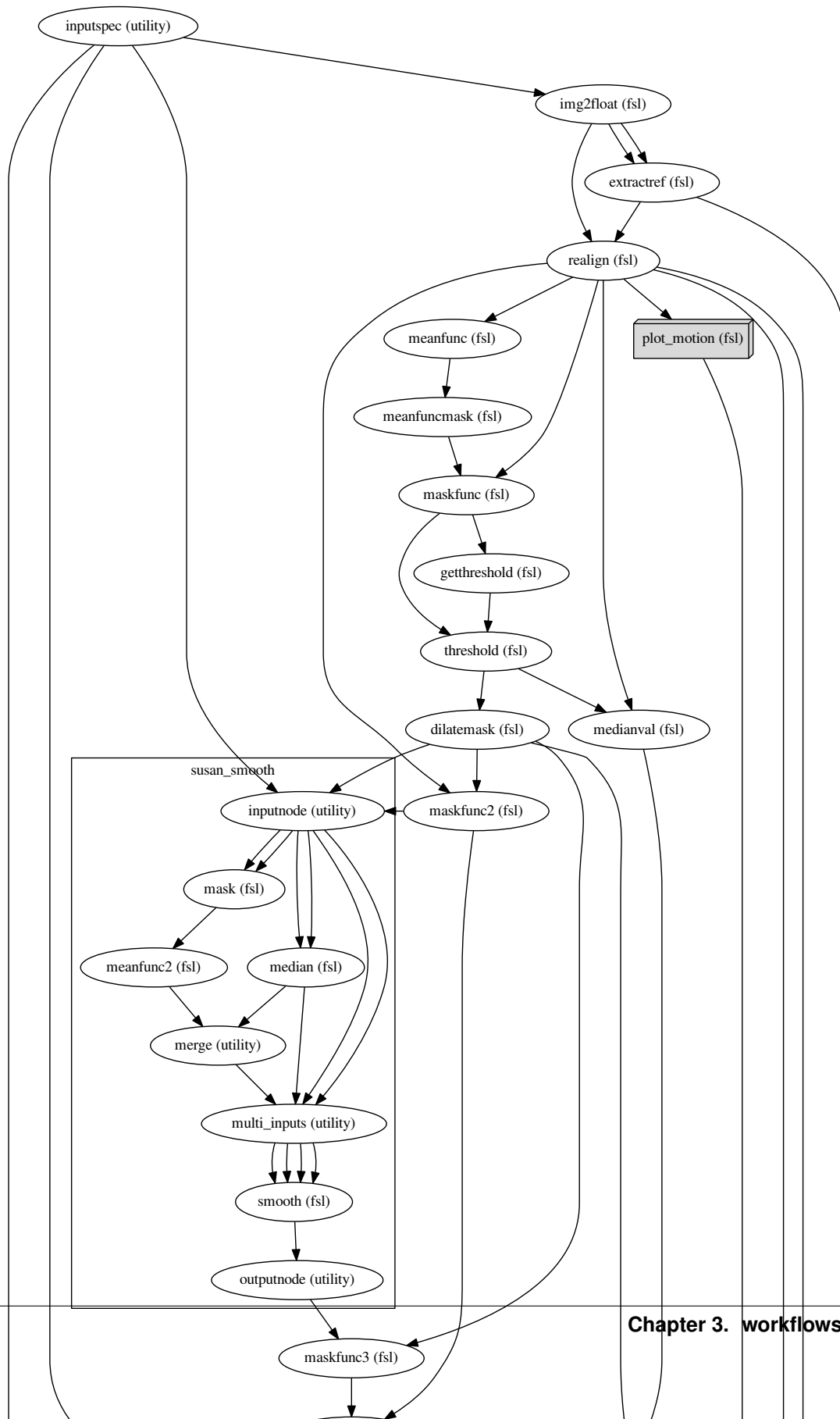
```
outputspec.reference : volume to which runs are realigned
outputspec.motion_parameters : motion correction parameters
outputspec.realigned_files : motion corrected files
outputspec.motion_plots : plots of motion correction parameters
outputspec.mask : mask file used to mask the brain
outputspec.smoothed_files : smoothed functional data
outputspec.highpassed_files : highpassed functional data (if highpass=True)
outputspec.mean : mean file
```

Example

```
>>> preproc = create_parallelfat_preproc()
>>> preproc.inputs.inputspec.func = ['f3.nii', 'f5.nii']
>>> preproc.inputs.inputspec.fwhm = 5
>>> preproc.inputs.inputspec.highpass = 128./(2*2.5)
>>> preproc.base_dir = '/tmp'
>>> preproc.run()
```

```
>>> preproc = create_parallelfat_preproc(highpass=False)
>>> preproc.inputs.inputspec.func = 'f3.nii'
>>> preproc.inputs.inputspec.fwhm = 5
>>> preproc.base_dir = '/tmp'
>>> preproc.run()
```


Graph



3.2.4 `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 (float or list of floats)
inputnode.mask_file : mask used for estimating SUSAN thresholds (but not for
↳smoothing)
```

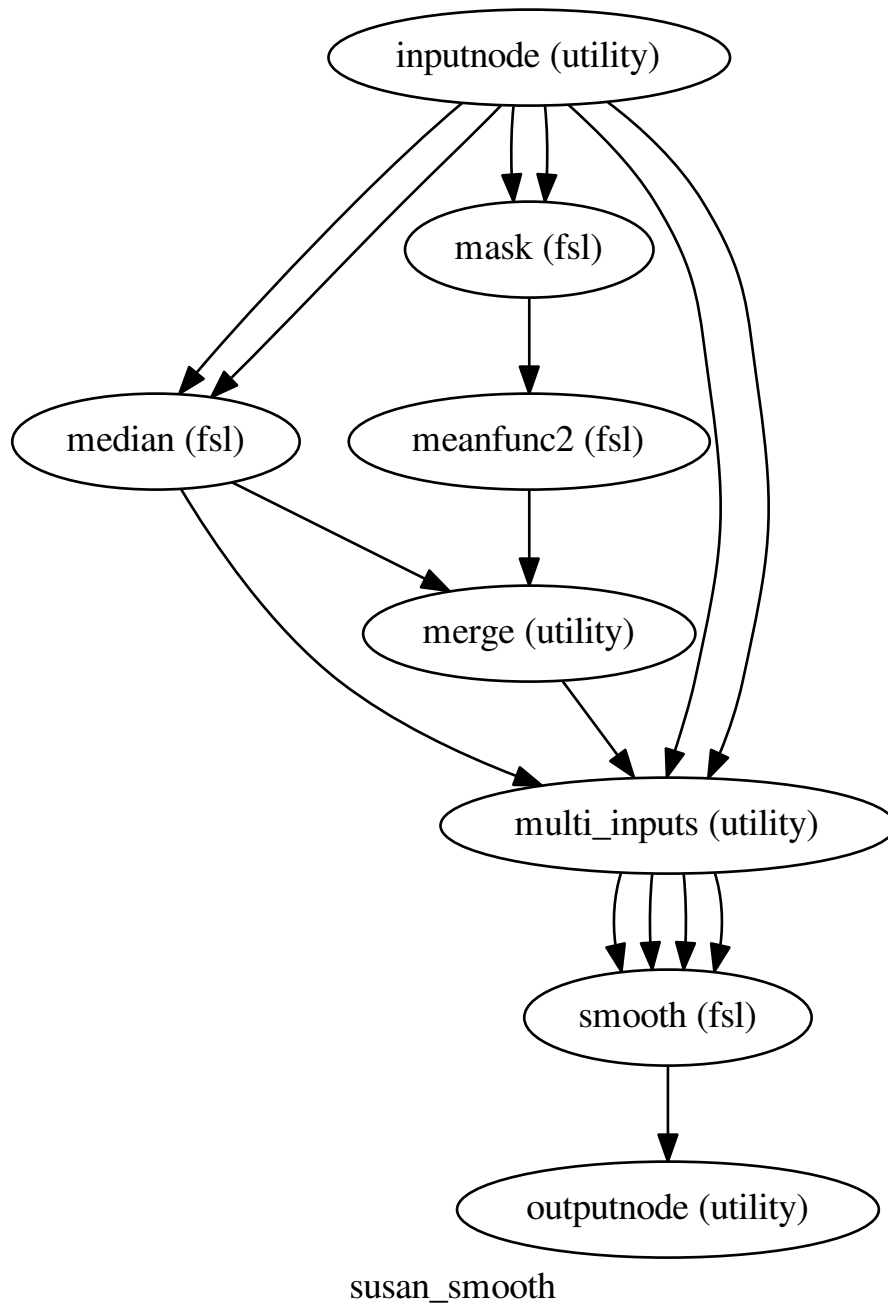
Outputs:

```
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



3.2.5 chooseindex()

[Link to code](#)

3.2.6 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

3.2.7 getbtthresh()

[Link to code](#)

3.2.8 getmeanscale()

[Link to code](#)

3.2.9 getthresshop()

[Link to code](#)

3.2.10 getusans()

[Link to code](#)

3.2.11 pickfirst()

[Link to code](#)

3.2.12 pickmiddle()

[Link to code](#)

3.2.13 pickrun()

[Link to code](#)

pick file from list of files

3.2.14 pickvol()

[Link to code](#)

3.3 workflows.fmri.spm.preprocess

3.3.1 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()
```

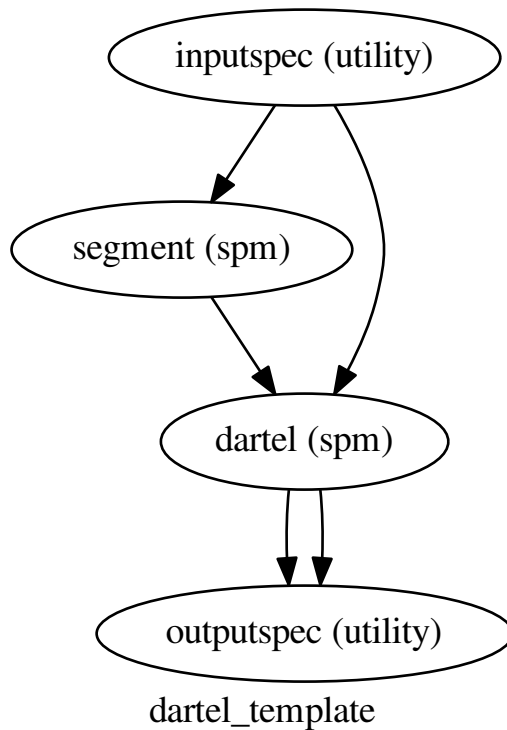
Inputs:

```
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



3.3.2 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

```

(continues on next page)

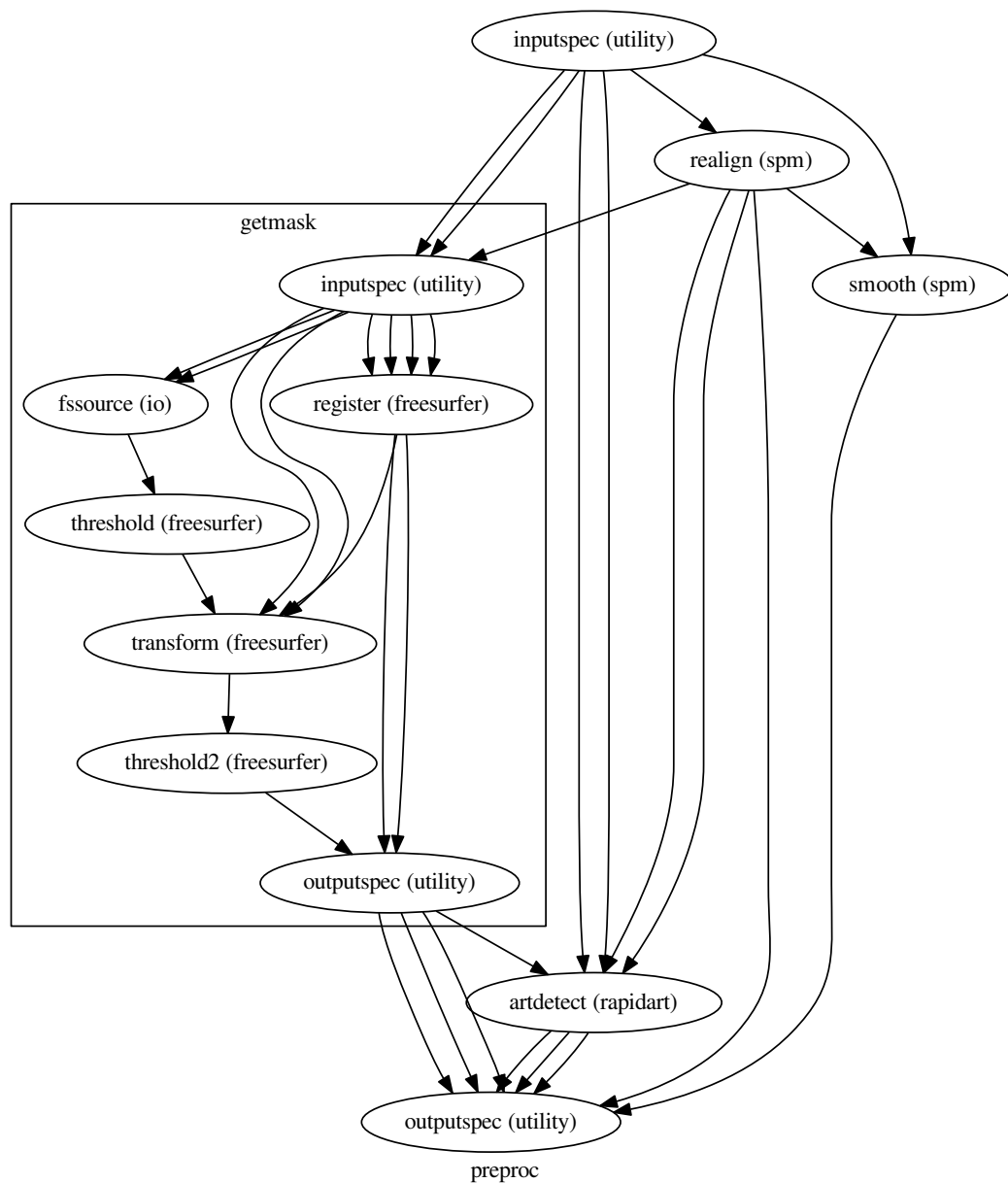
(continued from previous page)

```
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



3.3.3 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()
```

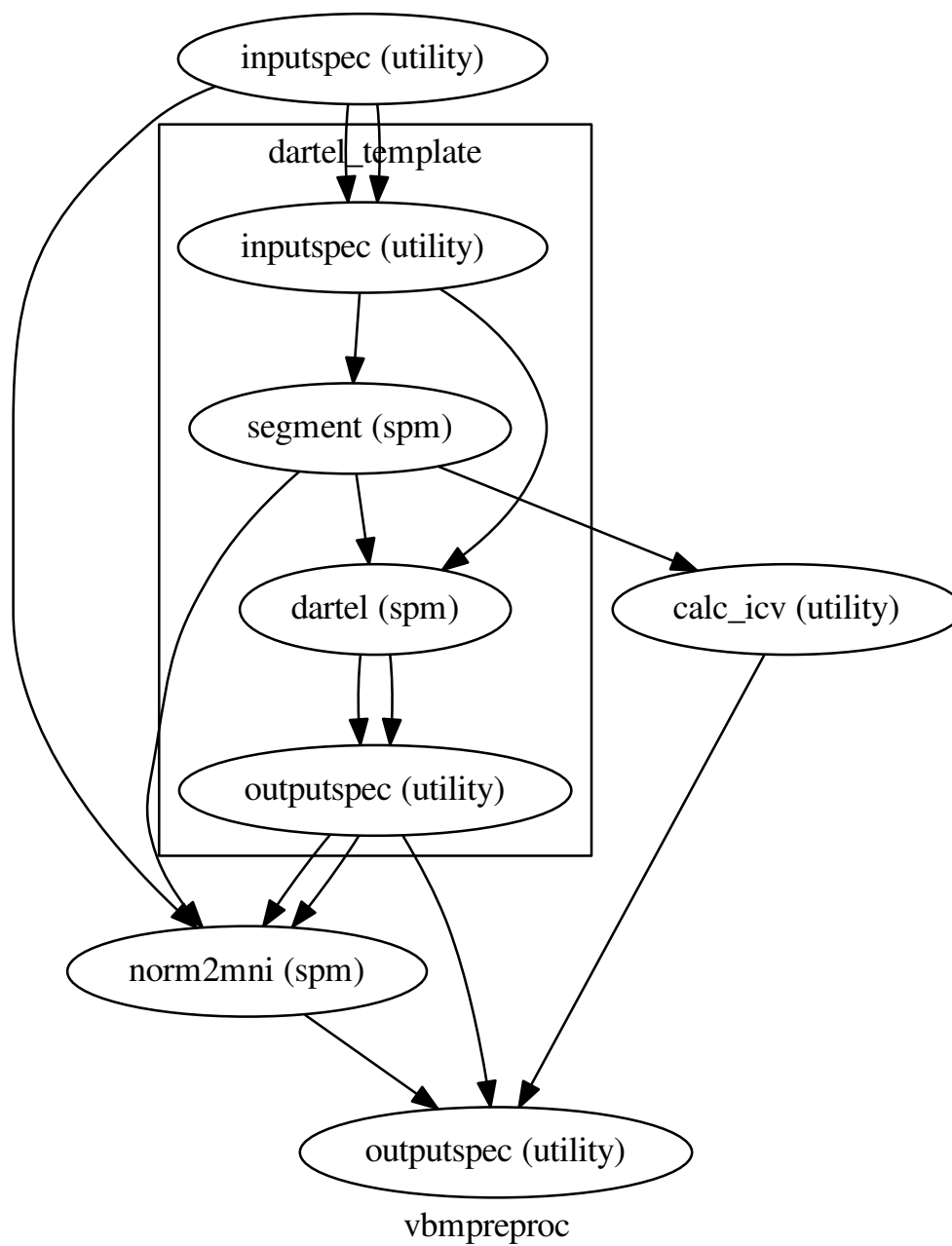
Inputs:

```
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



4.1 workflows.misc.utils

4.1.1 `get_affine()`

[Link to code](#)

4.1.2 `get_data_dims()`

[Link to code](#)

4.1.3 `get_vox_dims()`

[Link to code](#)

4.1.4 `id_list_from_lookup_table()`

[Link to code](#)

4.1.5 `region_list_from_volume()`

[Link to code](#)

4.1.6 `select_aparc()`

[Link to code](#)

4.1.7 `select_aparc_annot()`

[Link to code](#)

5.1 workflows.rsfmri.fsl.resting

5.1.1 create_realign_flow()

[Link to code](#)

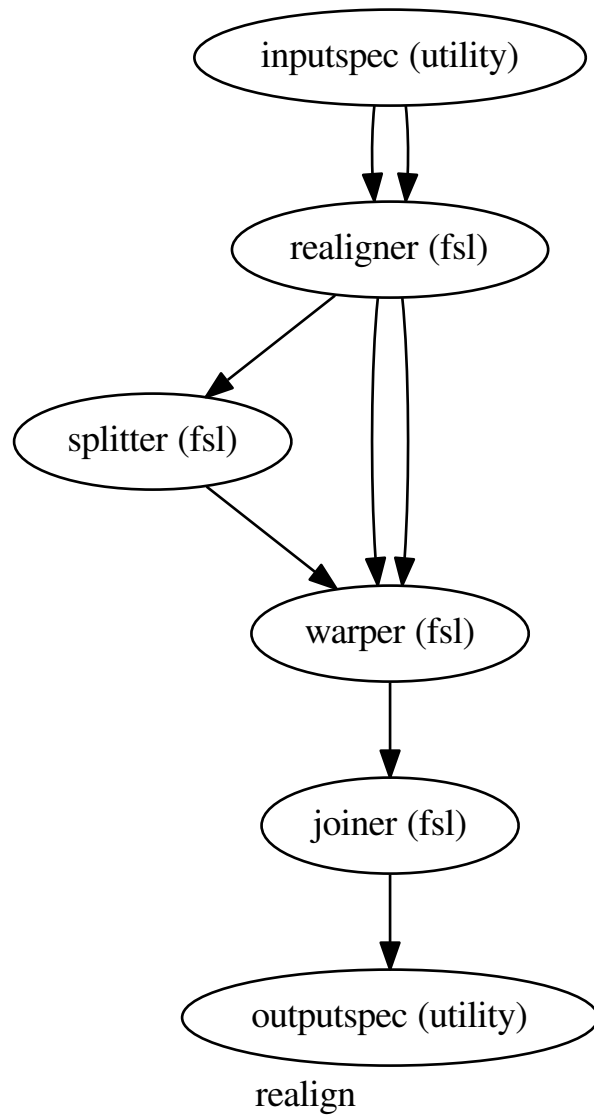
Realign a time series to the middle volume using spline interpolation

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



5.1.2 `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)
```

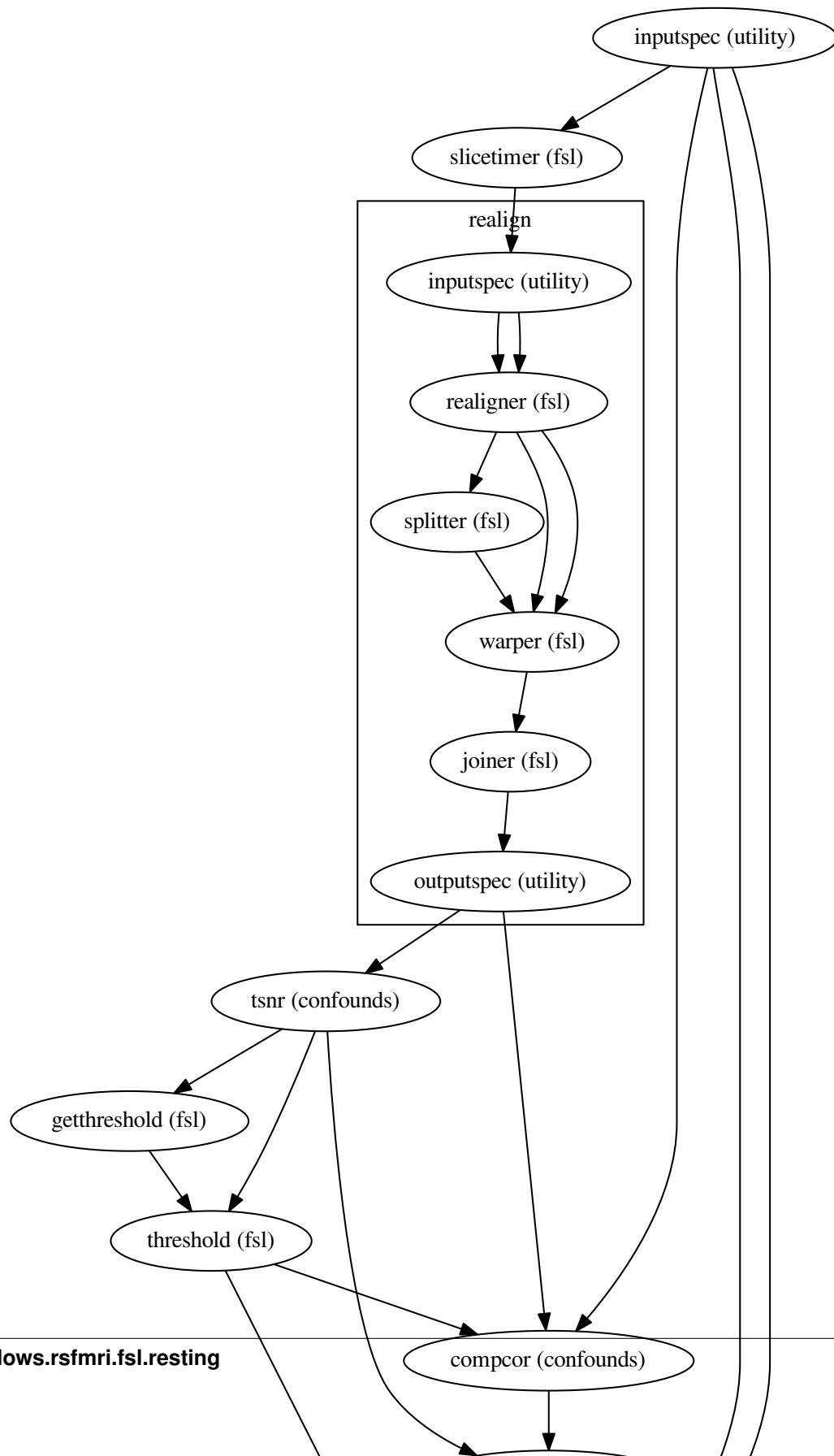
Outputs:

```
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



5.1.3 `select_volume()`

[Link to code](#)

Return the middle index of a file

6.1 workflows.smri.ants.ANTSBuildTemplate

6.1.1 ANTSTemplateBuildSingleIterationWF()

[Link to code](#)

Inputs:

```
inputspec.images :  
inputspec.fixed_image :  
inputspec.ListOfPassiveImagesDictionaries :
```

Outputs:

```
outputspec.template :  
outputspec.transforms_list :  
outputspec.passive_deformed_templates :
```


6.2 workflows.smri.ants.antsRegistrationBuildTemplate

6.2.1 antsRegistrationTemplateBuildSingleIterationWF()

[Link to code](#)

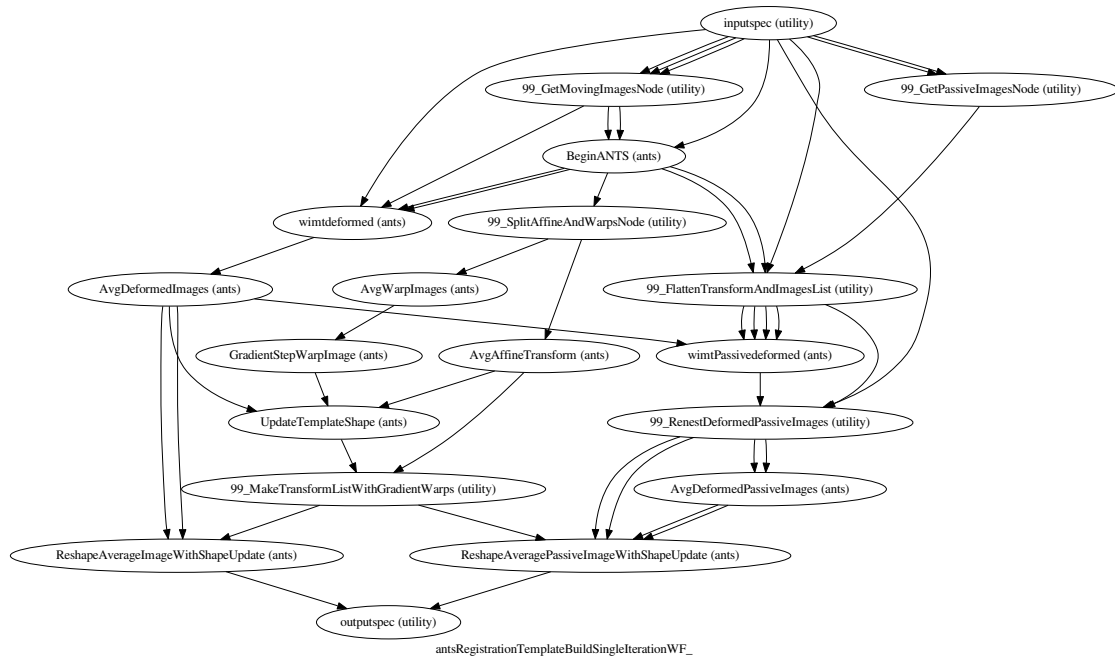
Inputs:

```
inputspec.images :
inputspec.fixed_image :
inputspec.ListOfPassiveImagesDictionaries :
inputspec.interpolationMapping :
```

Outputs:

```
outputspec.template :
outputspec.transforms_list :
outputspec.passive_deformed_templates :
```

Graph



6.2.2 FlattenTransformAndImagesList()

[Link to code](#)

6.2.3 GetFirstListElement()

[Link to code](#)

6.2.4 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.

6.2.5 GetPassiveImages()

[Link to code](#)

6.2.6 MakeTransformListWithGradientWarps()

[Link to code](#)

6.2.7 RenestDeformedPassiveImages()

[Link to code](#)

6.2.8 SplitAffineAndWarpComponents()

[Link to code](#)

6.2.9 makeListOfOneElement()

[Link to code](#)

6.3 workflows.smri.freesurfer.autorecon1

6.3.1 checkT1s()

[Link to code](#)

Verifying size of inputs and setting workflow parameters

6.3.2 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:

6.4 workflows.smri.freesurfer.autorecon2

6.4.1 copy_ltas()

[Link to code](#)

6.5 workflows.smri.freesurfer.bem

6.5.1 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()
```

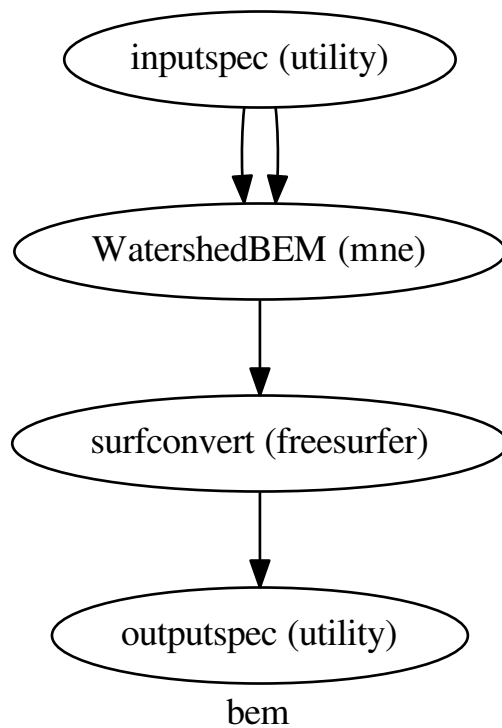
Inputs:

```
inputs.spec.subject_id : freesurfer subject id
inputs.spec.subjects_dir : freesurfer subjects directory
```

Outputs:

```
outputs.spec.meshes : output boundary element meshes in (by default)
                      stereolithographic (.stl) format
```

Graph



6.6 workflows.smri.freesurfer.recon

6.6.1 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.

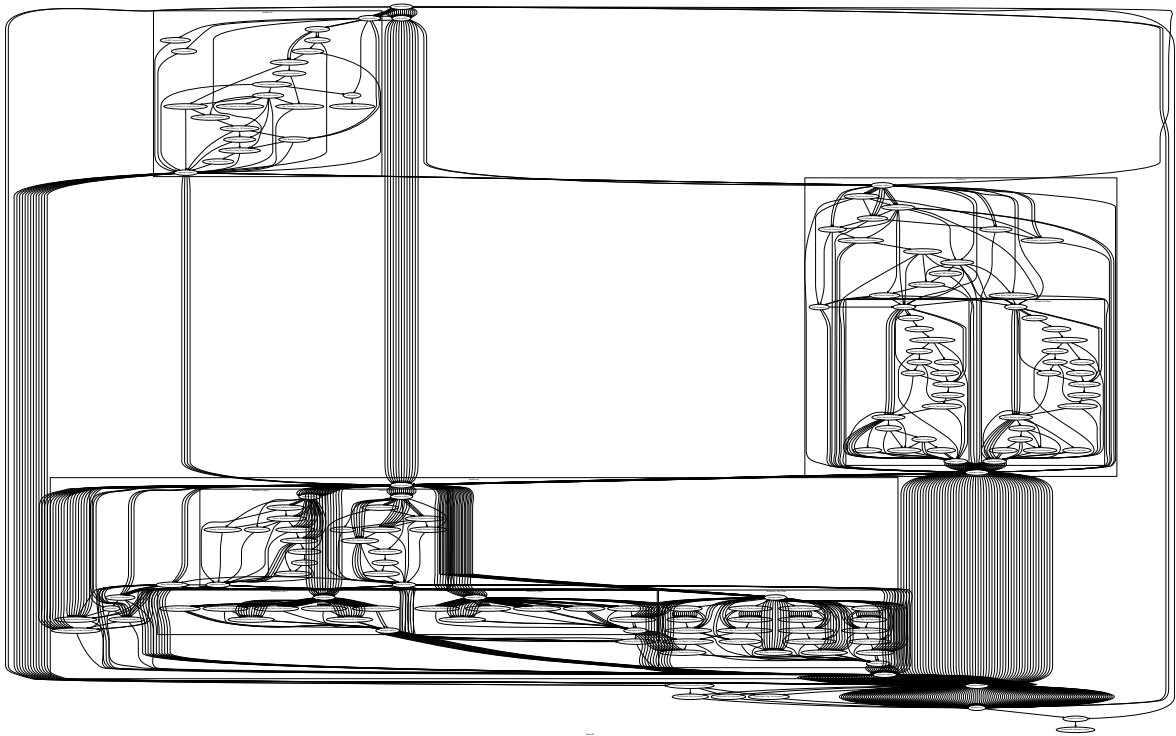
Example

```
>>> 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_all.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



6.6.2 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

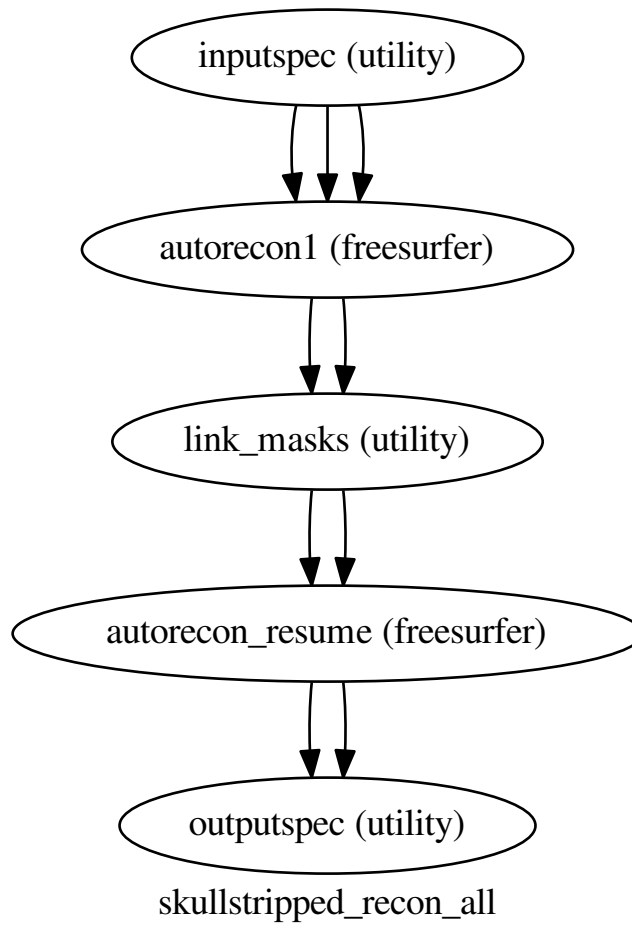
```
>>> from nipy.workflows.smri.freesurfer import create_skullstripped_recon_flow
>>> recon_flow = create_skullstripped_recon_flow()
>>> recon_flow.inputs.inputspec.subject_id = 'subj1'
>>> recon_flow.inputs.inputspec.T1_files = 'T1.nii.gz'
>>> recon_flow.run()
```

Inputs:: inputspec.T1_files : skullstripped T1_files (mandatory) inputspec.subject_id : freesurfer subject id (optional) inputspec.subjects_dir : freesurfer subjects directory (optional)

Outputs:

```
outputspec.subject_id : freesurfer subject id
outputspec.subjects_dir : freesurfer subjects directory
```

Graph



6.7 workflows.smri.freesurfer.utils

6.7.1 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

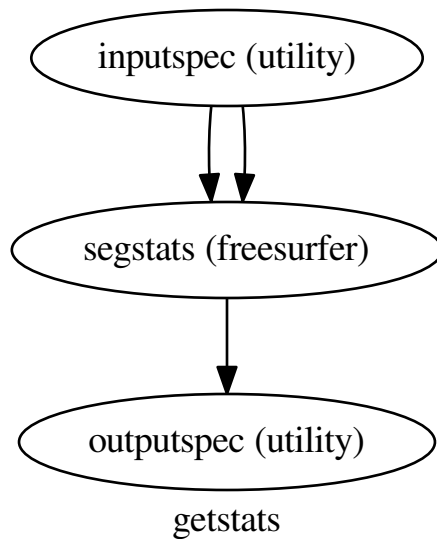
```

Outputs:

```

outputspec.stats_file : stats file

```

Graph**6.7.2 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'

```

(continues on next page)

(continued from previous page)

```
>>> getmask.inputs.inputs.spec.subject_id = 's1'
>>> getmask.inputs.inputs.spec.subjects_dir = '.'
>>> getmask.inputs.inputs.spec.contrast_type = 't2'
```

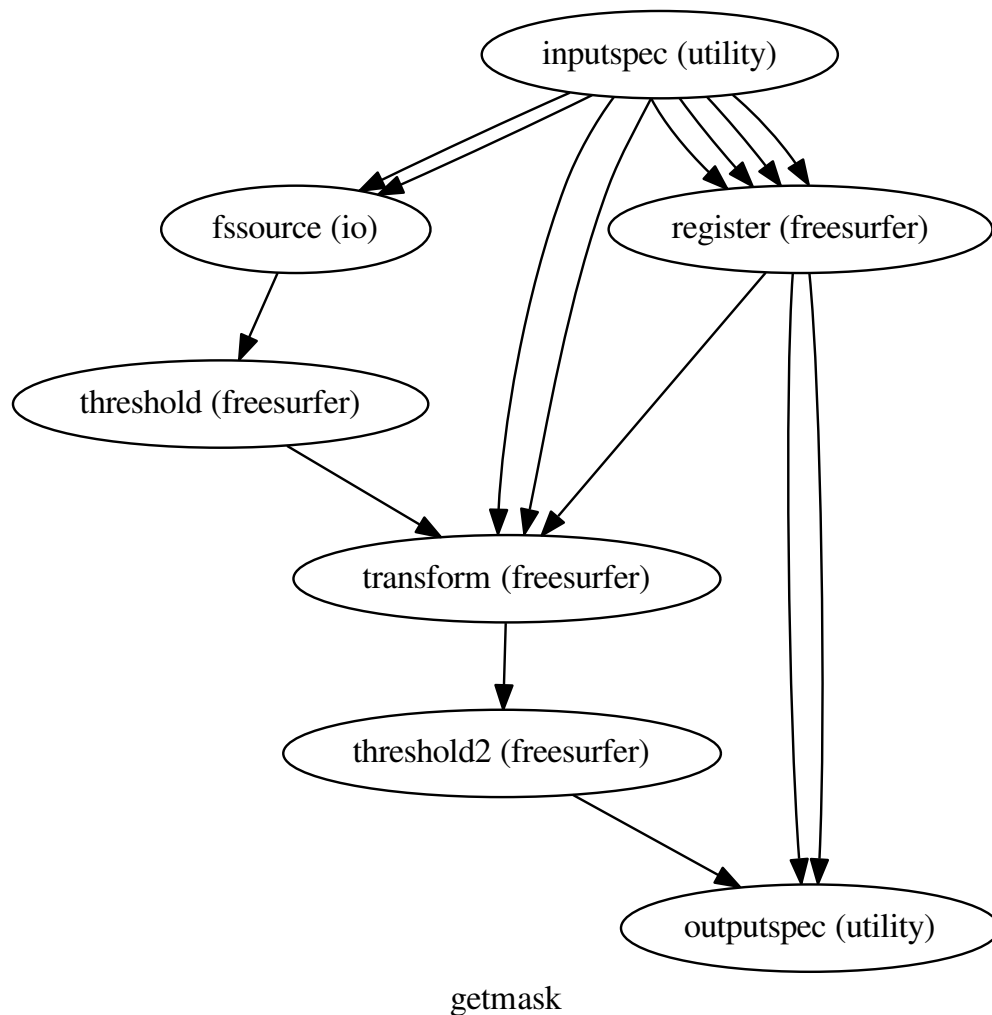
Inputs:

```
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



6.7.3 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 nipyte.workflows.smri.freesurfer import create_tessellation_flow
>>> tessflow = create_tessellation_flow()
>>> tessflow.inputs.inputs.spec.subject_id = 'subj1'
>>> tessflow.inputs.inputs.spec.subjects_dir = '.'

```

(continues on next page)

(continued from previous page)

```
>>> tessflow.inputs.inputspec.lookup_file = 'FreeSurferColorLUT.txt'
>>> tessflow.run()
```

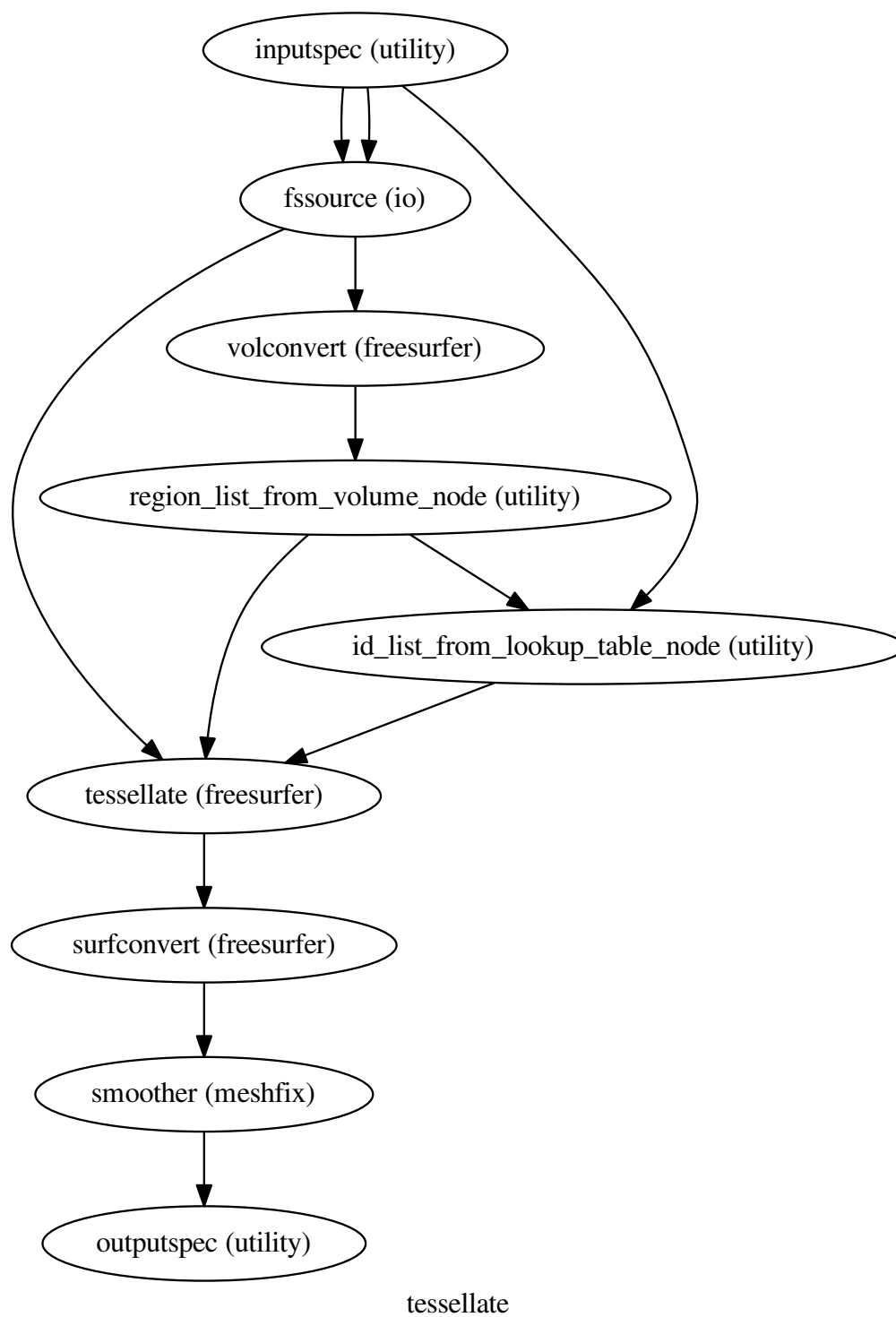
Inputs:

```
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



6.7.4 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.

6.7.5 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

6.7.6 get_aparc_aseg()

[Link to code](#)

Return the aparc+aseg.mgz file

6.7.7 getdefaultconfig()

[Link to code](#)

6.7.8 mkdir_p()

[Link to code](#)

6.8 workflows.smri.niftyreg.groupwise

6.8.1 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)
```

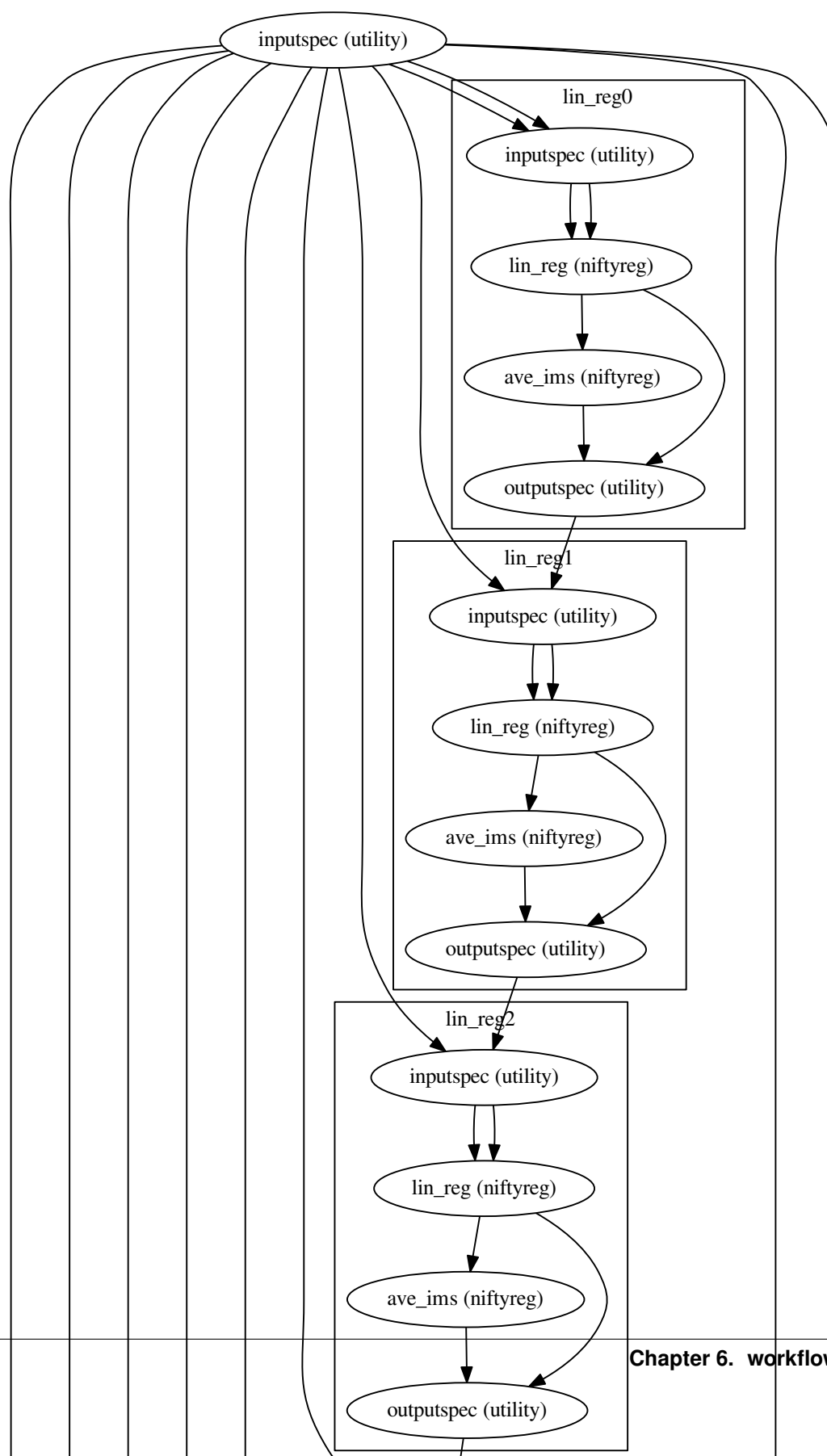
Outputs:

```
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



6.8.2 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
```

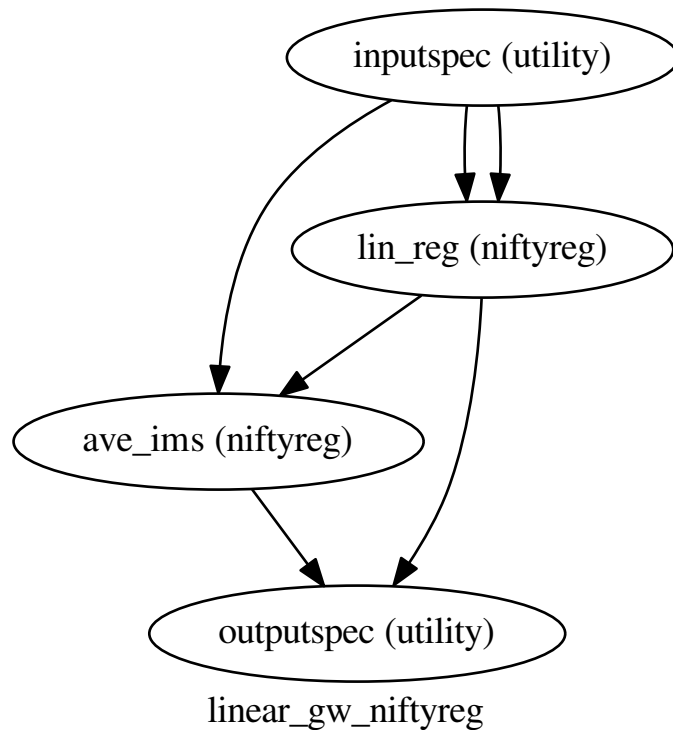
Optional arguments:

```
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



6.8.3 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
  
```

(continues on next page)

(continued from previous page)

```

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)

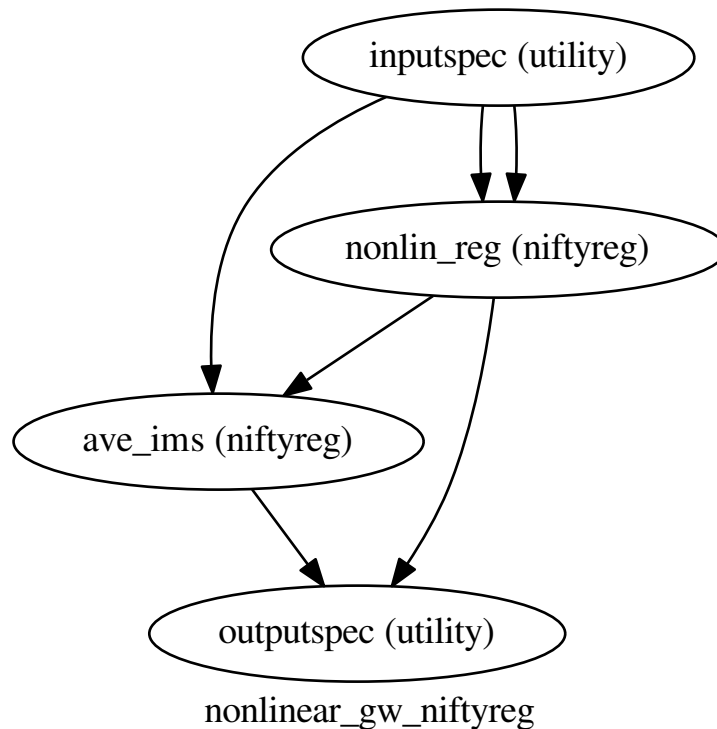
```

Example

```

>>> from nipy.workflows.smri.niftyreg import create_nonlinear_gw_step
>>> nlc = create_nonlinear_gw_step('nonlinear_coreg')
>>> nlc.inputs.inputs.spec.in_files = [
...     'file1.nii.gz', 'file2.nii.gz']
>>> nlc.inputs.inputs.spec.ref_file = ['ref.nii.gz']
>>> nlc.run()

```

Graph

- Examples

7.1 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
```

(continues on next page)

(continued from previous page)

```

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

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(
```

(continues on next page)

(continued from previous page)

```
interface=util.IdentityInterface(fields=["dwi", "bvecs", "bvals"]),
name="inputnode")
```

7.1.1 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 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

```
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']])])])
```

(continues on next page)

(continued from previous page)

```

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 exported 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

8.1 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 # pypeline engine
import nipype.interfaces.camino as camino
import nipype.interfaces.fsl as fsl
import nipype.interfaces.camino2trackvis as cam2trk
```

(continues on next page)

(continued from previous page)

```
import nipy.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

```
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 infosource 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=camino2trk.Camino2Trackvis(), name="camino2trk")
camino2trackvis.inputs.min_length = 30
camino2trackvis.inputs.voxel_order = 'LAS'
trk2camino = pe.Node(interface=camino2trk.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')
```

(continues on next page)

(continued from previous page)

```
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 Nipyne 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')])])
```

(continues on next page)

(continued from previous page)

```

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")])])

```

(continues on next page)

(continued from previous page)

```
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 datagrabbing 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

9.1 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!

9.2 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
```

(continues on next page)

(continued from previous page)

```

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.FSLCommand.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')

```

(continues on next page)

(continued from previous page)

```

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'

```

9.3 Creating the workflow's nodes

9.3.1 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.MRConvert(), 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')

```

9.3.2 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.

```
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.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.MRTrk2TrackVis(), name='tck2trk')
trk2tdi = pe.Node(interface=dipy.TrackDensityMap(), name='trk2tdi')
```

9.3.3 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']
```

(continues on next page)

(continued from previous page)

```
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
```

9.4 Connecting the workflow

Here we connect our processing pipeline.

9.4.1 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") ])])
```

(continues on next page)

(continued from previous page)

```
mapping.connect([(inputnode, tessflow, [("subjects_dir",
                                         "inputspec.subjects_dir")])])
mapping.connect([(inputnode, tessflow, [("subject_id",
                                         "inputspec.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']])])
```

9.4.2 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")])])
```

(continues on next page)

(continued from previous page)

```
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")])])
```

(continues on next page)

(continued from previous page)

```
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")])])
```

9.4.3 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")])])
```

(continues on next page)

(continued from previous page)

```
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")])])
```

9.4.4 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')
```

(continues on next page)

(continued from previous page)

```
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 `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 # pipeline 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('IPython', '0.10', 'tutorial1')
```

10.1 Setting up workflows

This is a generic workflow for DTI data analysis using the FSL

10.2 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', 'dwi']],
    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 `nipyype.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipyype.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
```

10.3 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')])])
```

10.4 Setup for Tracktography

Here we will create a workflow to enable deterministic tracktography

```
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('dtireresults')
```

```
def getstripdir(subject_id):
    return os.path.join(
        os.path.abspath('data/workingdir/dwiproc'),
        '_subject_id_%s' % subject_id)
```

10.5 Setup the pipeline that combines the 2 workflows: tractography & 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'),
```

(continues on next page)

(continued from previous page)

```
        ('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 # pipeline 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', 'tutorial11')
package_check('scipy', '0.7', 'tutorial11')
package_check('IPython', '0.10', 'tutorial11')
```

11.1 Setting up workflows

This is a generic workflow for DTI data analysis using the FSL

11.2 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
```

11.3 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') ])])
```

11.4 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') ])])
```

11.5 Setup the pipeline that combines the 2 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
```

(continues on next page)

(continued from previous page)

```
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 12

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 # pipeline 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('IPython', '0.10', 'tutorial1')
```

12.1 Setting up workflows

This is a generic workflow for DTI data analysis using the FSL

12.2 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 `nipyne.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipyne.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
```

12.3 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')])])
```

12.4 Setup for Tractography

Here we will create a workflow to enable probabilistic tractography 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 tractography

```
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)
```

12.5 Setup the pipeline that combines the 2 workflows: tractography & 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 Nipype source distribution under the examples directory.

dmRI: Group connectivity - Camino, FSL, FreeSurfer

13.1 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 *dmRI: 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.

13.1.1 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.camino.group_connectivity import create_group_
↳connectivity_pipeline
from nipy.workflows.dmri.connectivity.group_connectivity import (
    create_merge_networks_by_group_workflow,
    create_merge_group_networks_workflow,
    create_average_networks_by_group_workflow)
```

13.1.2 Set the proper directories

First, we import the necessary modules from nipy.

```
fs_dir = op.abspath('/usr/local/freesurfer')
subjects_dir = op.abspath('groupcondatapackage/subjects/')
data_dir = op.abspath('groupcondatapackage/data/')
fs.FSCommand.set_default_subjects_dir(subjects_dir)
fsl.FSLCommand.set_default_output_type('NIFTI')
```

13.1.3 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')
```

13.2 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:

- `nipy/workflows/dmri/mrtrix/group_connectivity.py`
- `nipy/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 Nipype source distribution under the examples directory.

dmRI: Group connectivity - MRtrix, FSL, FreeSurfer

14.1 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.

14.1.1 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)
```

14.1.2 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.FSCommand.set_default_subjects_dir(subjects_dir)
fsl.FSLCommand.set_default_output_type('NIFTI')
```

14.1.3 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')
```

14.2 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 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']])

    l1pipeline = 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
    l1pipeline.inputs.connectivity.mapping.fsl2mrtrix.invert_y = True
    l1pipeline.inputs.connectivity.mapping.csdeconv.maximum_harmonic_order = 6

    # Here we define the parcellation scheme and the number of tracks to produce
    parcellation_name = 'scale500'
    l1pipeline.inputs.connectivity.mapping.Parcellate.parcellation_name = _
    ↪parcellation_name
    cmp_config = cmp.configuration.PipelineConfiguration()
    cmp_config.parcellation_scheme = "Lausanne2008"
    l1pipeline.inputs.connectivity.mapping.inputnode_within.resolution_network_
    ↪file = cmp_config._get_lausanne_parcellation(
        'Lausanne2008')[parcellation_name]['node_information_graphml']
    l1pipeline.inputs.connectivity.mapping.probCSDstreamtrack.desired_number_of_
    ↪tracks = 100000

    l1pipeline.run()
    l1pipeline.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()

```

(continues on next page)

(continued from previous page)

```
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.

15.1 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']],
```

(continues on next page)

(continued from previous page)

```
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")
```

15.1.1 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.MRTrk2TrackVis(), name='tck2trk')

```

15.1.2 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")])])

```

(continues on next page)

(continued from previous page)

```
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")])])
```

(continues on next page)

(continued from previous page)

```
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 Nipype source distribution under the `examples` directory.

16.1 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', 'dwidata']],
    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,
```

(continues on next page)

(continued from previous page)

```
# 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_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

```
inputnode = pe.Node(
    niu.IdentityInterface(fields=["dwi", "bvecs", "bvals", "dwi_rev"]),
    name="inputnode")
```

16.2 Setup for dMRI preprocessing

In this section we initialize the appropriate workflow for preprocessing of diffusion images.

16.2.1 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)
```

16.2.2 Bias field correction

Finally, we set up a node to correct for a single multiplicative bias field from computed on the b_0 image, as suggested in [Jeurissen2014].

```
bias = remove_bias()
```

16.3 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'
```

(continues on next page)

(continued from previous page)

```

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), '11_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), '12')
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

```

(continues on next page)

(continued from previous page)

```
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
```

This workflow also requires 2mm subcortical templates that are available from [MindBoggle](#). Specifically the 2mm version of the [MNI template](#).

Import necessary modules from nipy.

```
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, CalculateMedian
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())
```

(continues on next page)

(continued from previous page)

```

    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 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

    Example
    -----
        See code below
    """

    register = pe.Workflow(name=name)

    inputnode = pe.Node(
        interface=niu.IdentityInterface(fields=[
            'source_files', 'mean_image', 'anatomical_image', 'target_image',
            'target_image_brain', 'config_file'
        ]),
        name='inputspec')
    outputnode = pe.Node(
        interface=niu.IdentityInterface(fields=[
            'func2anat_transform', 'anat2target_transform',
            'transformed_files', 'transformed_mean', 'anat2target',
            'mean2anat_mask'
        ]),
        name='outputspec')
    """

```

(continues on next page)

(continued from previous page)

```
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(binarize, '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')
```

```
"""
```

(continues on next page)

(continued from previous page)

```

Compute registration between the subject's structural and MNI template

* All parameters are set using the example from:
#https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh
* 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.
"""

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

```

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.terminal_output = 'file'

```

(continues on next page)

(continued from previous page)

```

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.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

```

(continues on next page)

(continued from previous page)

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

See code below

"""

```

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.MRIConvert(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(bbregister, 'registered_file', mean2anat_mask, 'in_file')
"""

```

use aparc+aseg's brain mask

"""

(continues on next page)

(continued from previous page)

```

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(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

    * All parameters are set using the example from:
      #https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh
    * 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.
"""

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]]

```

(continues on next page)

(continued from previous page)

```

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

```

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.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.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')

```

(continues on next page)

(continued from previous page)

```

register.connect(inputnode, 'source_files', warppall, 'input_image')
register.connect(merge, 'out', warppall, '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(bbregister, 'out_reg_file', outputnode, 'out_reg_file')
register.connect(bbregister, '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)

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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:

```

(continues on next page)

(continued from previous page)

```

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')]),

```

(continues on next page)

(continued from previous page)

```

])

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'),

```

(continues on next page)

(continued from previous page)

```

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(CalculateMedian(), name='median')

```

(continues on next page)

(continued from previous page)

```

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')

```

(continues on next page)

(continued from previous page)

```

        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, 'inputs.spec.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, 'outputs.spec.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, 'outputs.spec.aparc', get_roi_mean,
            'segmentation_file')

        get_roi_tsnr = pe.MapNode(

```

(continues on next page)

(continued from previous page)

```

        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_bbreg_brain_mask', 'median_brain_mask')
    return subs

subsgen = pe.Node(
    niu.Function(
        input_names=[
            'subject_id', 'conds', 'run_id', 'model_id', 'task_id'
        ],

```

(continues on next page)

(continued from previous page)

```

        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
"""

```

(continues on next page)

(continued from previous page)

```

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',

```

(continues on next page)

(continued from previous page)

```

        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 Nipype 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.

19.1 Preparing environment

19.1.1 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.

19.1.2 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 ..
```

19.2 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
```

(continues on next page)

(continued from previous page)

```
import nipyne.interfaces.utility as util # utility
import nipyne.pipeline.engine as pe # pypeline engine
```

19.2.1 iminaries

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

```
# 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')
```

19.2.2 eprocessing workflow

```
preproc = pe.Workflow(name='preproc')
```

Use `nipyne.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 `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(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 `nipyne.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 `nipyne.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 `nipyne.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 smoothin. using `nipy.interfaces.spm.Smooth`. In addition, we use a smoothing routine from freesurfer (`nipy.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')]),
])
```

19.2.3 Set up volume analysis workflow

```
volanalysis = pe.Workflow(name='volanalysis')
```

Generate SPM-specific design information using `nipy.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 `nipy.interfaces.spm.Level1Design`.

```
levelldesign = pe.Node(interface=spm.Level1Design(), name="levelldesign")
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 `nipyype.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, levellestimate, [('spm_mat_file', 'spm_mat_file')]),
    (levellestimate, contrastestimate,
     [('spm_mat_file', 'spm_mat_file'), ('beta_images', 'beta_images'),
      ('residual_image', 'residual_image')]),
])
```

19.2.4 Set up surface analysis workflow

We simply clone the volume analysis workflow.

```
surfanalysis = volanalysis.clone(name='surfanalysis')
```

19.2.5 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 `nipyype.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 `nipyype.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 `nipyype.interfaces.freesurfer.ApplyVolTransform` to convert contrast images into freesurfer space.

```
normwreg = pe.MapNode(
    interface=fs.ApplyVolTransform(),
    iterfield=['source_file'],
    name='applyreg2con')
```

Use `nipyype.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')]),
])
```

19.2.6 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', 'realigned.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, [
        ('realigned.realignment_parameters', 'modelspec.realignment_parameters'),
        ('volsmooth.smoothed_files', 'modelspec.functional_runs'),
        ('art.outlier_files', 'modelspec.outlier_files'),
        ('threshold.binary_file', 'levelldesign.mask_image')]),
    (preproc, surfanalysis, [
        ('realigned.realignment_parameters', 'modelspec.realignment_parameters'),
        ('surfsmooth.smoothed_file', 'modelspec.functional_runs'),
        ('art.outlier_files', 'modelspec.outlier_files'),
        ('threshold.binary_file', 'levelldesign.mask_image')])])

# attach volume contrast normalization components
l1pipeline.connect([
    (preproc, volnorm, [
        ('fssource.orig', 'convert2nii.in_file'),
        ('surfregister.out_reg_file', 'applyreg2con.reg_file'),
        ('fssource.orig', 'applyreg2con.target_file')]),
    (volanalysis, volnorm, [
        ('contrastestimate.con_images',
```

(continues on next page)

(continued from previous page)

```
        'convertimg2nii.in_file'),
    ]))
```

The nipyne 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 mical 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

```
# 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 `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.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

19.2.7 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
```

19.2.8 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]
```

19.2.9 Set up node specific inputs

We replicate the modelspec parameters separately for the surface- and volume-based analysis.

```
modelspecref = llpipeline.inputs.volanalysis.modelspec
modelspecref.input_units = 'secs'
modelspecref.time_repetition = 3.
modelspecref.high_pass_filter_cutoff = 120

modelspecref = llpipeline.inputs.surfanalysis.modelspec
modelspecref.input_units = 'secs'
modelspecref.time_repetition = 3.
modelspecref.high_pass_filter_cutoff = 120

l1designref = llpipeline.inputs.volanalysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition

l1designref = llpipeline.inputs.surfanalysis.level1design
l1designref.timing_units = modelspecref.output_units
l1designref.interscan_interval = modelspecref.time_repetition

llpipeline.inputs.inputnode.contrasts = contrasts
```

19.2.10 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.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')]),
])
```

19.2.11 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) rkflow.

```
if __name__ == '__main__':
    level1.run()
    level1.write_graph(graph2use='flat')
```

19.2.12 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 Nipy 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
```

20.1 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')
```

20.2 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.

20.3 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'])
```

(continues on next page)

(continued from previous page)

```
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 getthresshop(thresh):
    return '-thr %.10f -Tmin -bin' % (0.1 * thresh[0][1])

preproc.connect(getthresh, ('out_stat', getthresshop), 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 `nipype.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")
```

(continues on next page)

(continued from previous page)

```

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')]),
    (dilatmask, art, [(['out_file', 'mask_file')]),
])

```

20.4 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')]),
])

```

20.5 Set up fixed-effects workflow

```
fixed_fx = pe.Workflow(name='fixedfx')
```

Use `nipyre.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 `nipyre.interfaces.fsl.L2Model` to generate subject and condition specific level 2 model design files

```
level2model = pe.Node(interface=fsl.L2Model(), name='l2model')
```

Use `nipyre.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')]),
])
```

20.6 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'),
```

(continues on next page)

(continued from previous page)

```

        ('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.varcopes', sort_copes), 'varcopemerge.in_files'),
        (('conestimate.copes', num_copes), 'l2model.num_copes'),
    ]))

```

20.7 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'.

```

# 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.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 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')

```

(continues on next page)

(continued from previous page)

```
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 `nipy.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 nipy.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
```

20.7.1 Set up complete workflow

```
l1pipeline = pe.Workflow(name="level1")
l1pipeline.base_dir = os.path.abspath('./fsl/workingdir')
l1pipeline.config = {
    "execution": {
```

(continues on next page)

(continued from previous page)

```
        "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'),
    ]),
])
```

20.8 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 `nipype.pipeline.engine.Pipeline.Run` function needs to be called.

```
if __name__ == '__main__':
    lllpipeline.write_graph()
    outgraph = lllpipeline.run()
    # lllpipeline.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.

CHAPTER 21

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 # system functions
from nipy.interfaces import io as nio # Data i/o
from nipy.interfaces import utility as niu # Utilities
from nipy.interfaces import fsl # fsl
from nipy.pipeline import engine as pe # pipeline engine
from nipy.algorithms import modelgen as model # model generation
from nipy.workflows.fmri.fsl import (
    create_featreg_preproc, create_modelfit_workflow, create_reg_workflow)
from nipy.interfaces.base import Bunch
```

21.1 iminaries

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')
```

21.2 Experiment specific components

This tutorial does a single subject analysis so we are not using infosource 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
```

(continues on next page)

(continued from previous page)

```
# Map field names to individual subject runs.
info = dict(func=[['fmri']], struct=[['structural']])
```

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.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'
```

21.2.1 Set up complete workflow

```

llpipeline = pe.Workflow(name="level1")
llpipeline.base_dir = os.path.abspath('./fsl_feeds/workingdir')
llpipeline.config = {
    "execution": {
        "crashdump_dir": os.path.abspath('./fsl_feeds/crashdumps')
    }
}

llpipeline.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')

```

21.3 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 Nipype 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)
```

22.1 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')

level1_workflow = pe.Workflow(name='level1flow')

preproc = create_featreg_preproc(whichvol='first')

modelfit = create_modelfit_workflow()
```

(continues on next page)

(continued from previous page)

```
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")

level1_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')])])
```

22.2 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]

level1_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'),
```

(continues on next page)

(continued from previous page)

```
((('outputspec.varcopes', sort_copes), 'inputspec.varcopes'),
 (('outputspec.copes', num_copes), 'l2model.num_copes'),
 ]))
```

22.3 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 `nipyype.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', 'inputspec.func')]),
])
```

22.4 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()
```

(continues on next page)

(continued from previous page)

```
level1_workflow.run()  
# level1_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 # pipeline engine
import nipy.algorithms.rapidart as ra # artifact detection
import nipy.algorithms.modelgen as model # model specification
import os # system functions
```

23.1 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

```
# 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)
```

23.2 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

compute_mask = pe.Node(interface=ComputeMask(), name="compute_mask")
```

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'
```

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'
```

Smooth the functional data using `nipyne.interfaces.spm.Smooth`.

```
smooth = pe.Node(interface=spm.Smooth(), name="smooth")
smooth.inputs.fwhm = 4
```

23.3 Set up analysis 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]
```

Generate design information using `nipyne.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]

```

23.4 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") ]))

```

(continues on next page)

(continued from previous page)

```
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
```

24.1 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('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')
```

(continues on next page)

(continued from previous page)

```
# 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)
```

24.2 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 Nipyte source distribution under the `examples` directory.

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 nipy tutorial directory:

```
python fmri_spm.py
```

Import necessary modules from nipy.

```
from __future__ import print_function
from builtins import str
from builtins import range

import os # system functions

from nipy import config
# config.enable_provenance()

from nipy.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 nipy.interfaces.io as nio # Data i/o
import nipy.interfaces.utility as util # utility
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 nipy.interfaces.matlab as mlab
```

25.1 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

```
# 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)
```

25.2 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 `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(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 `nipyne.interfaces.fsl.BET`.

```
skullstrip = pe.Node(interface=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(interface=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(interface=spm.Normalize(), name="normalize")
normalize.inputs.template = os.path.abspath('data/T1.nii')
```

Smooth the functional data using `nipyne.interfaces.spm.Smooth`.

```
smooth = pe.Node(interface=spm.Smooth(), name="smooth")
fwhmlist = [4]
smooth.iterables = ('fwhm', fwhmlist)
```

25.3 Set up analysis 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]
```

Generate SPM-specific design information using `nipyype.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 `nipyype.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 `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}
```

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")
contrastestimate.inputs.contrasts = contrasts
contrastestimate.overwrite = True
contrastestimate.config = {'execution': {'remove_unnecessary_outputs': False}}
```

25.4 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.

```
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')]),
```

(continues on next page)

(continued from previous page)

```

(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')]),
])

```

25.5 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(interface=nio.DataSink(), name="datasink")
datasink.inputs.base_directory = os.path.abspath('spm_tutorial/l1output')

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')]),

```

(continues on next page)

(continued from previous page)

```
l1)
```

Use `nipyype.interfaces.io.DataGrabber` to extract the contrast ges across a group of first level subjects. Unlike the previous pipeline that iterated over subjects, this pipeline will iterate over

```
# 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 `nipyype.interfaces.spm.OneSampleTTestDesign` to perform a simple statistical analysis of the contrasts from the group of n this example).

```
# setup a 1-sample t-test node
onesampltestdes = 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 lettest).

```
l2pipeline = pe.Workflow(name="level2")
l2pipeline.base_dir = os.path.abspath('spm_tutorial/l2output')
l2pipeline.connect([
    (l2source, onesampltestdes, [('outfiles', 'in_files')]),
    (onesampltestdes, l2estimate, [('spm_mat_file', 'spm_mat_file')]),
    (l2estimate, l2conestimate,
     [('spm_mat_file', 'spm_mat_file'), ('beta_images', 'beta_images'),
      ('residual_image', 'residual_image')]),
])
```

25.6 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__':
    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

26.1 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.utility as util # utility
import nipype.pipeline.engine as pe # pypeline engine
import nipype.algorithms.modelgen as model # model specification
import os # system functions
```

26.1.1 Preliminaries

```
# Set the way matlab should be called
mlab.MatlabCommand.set_default_matlab_cmd("matlab -nodesktop -nosplash")
```

26.1.2 IOWs

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. `eprocessing` 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 `nipyype.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 `nipyype.interfaces.spm.Realign` for motion correction and register all images to the mean image.

```
realign = pe.Node(interface=spm.Realign(), name="realign")
```

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")
```

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 emplate 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 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')]),
```

(continues on next page)

(continued from previous page)

```

(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')]),
])

```

26.1.3 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")
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}

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")

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')]),
])

```

26.1.4 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:
    llpipeline.connect([(preproc, llanalysis,
                        [(['smooth.smoothed_files',
                          'modelspec.functional_runs'])])])
else:
    def makelist(item):
        return [item]

    llpipeline.connect([(preproc, llanalysis,
                        [(['smooth.smoothed_files', makelist),
                          'modelspec.functional_runs'])])])

```

26.1.5 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 `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%s_%03d.img'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True

```

26.1.6 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
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
```

26.1.7 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_auditory_tutorial/workingdir')

level1.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                (datasource, llpipeline, [('struct',
                                             'preproc.coregister.source')])])

if merge_to_4d:
    level1.connect([(datasource, llpipeline, [('func',
                                             'preproc.merge.in_files')])])
```

(continues on next page)

(continued from previous page)

```

else:
    levell.connect([(datasource, llpipeline, [('func',
                                              'preproc.realign.in_files')])])

```

26.1.8 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')]),
    (llpipeline, datasink,
     [('analysis.contrastestimate.con_images', 'contrasts.@con'),
      ('analysis.contrastestimate.spmT_images', 'contrasts.@T')]),
])

code discussed above sets up all the necessary data structures

```

opriate parameters and the connectivity between the sses, but does not generate any output. To actually run the sis on the data the `nipyype.pipeline.engine.Pipeline.Run`

```

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 Nipyype 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 nipyre tutorial directory:

```
python fmri_spm_dartel.py
```

Import necessary modules from nipyre.

```
from __future__ import print_function
from builtins import str
from builtins import range

import nipyre.interfaces.io as nio # Data i/o
import nipyre.interfaces.spm as spm # spm
import nipyre.workflows.fmri.spm as spm_wf # spm
import nipyre.interfaces.fsl as fsl # fsl
from nipyre.interfaces import utility as niu # Utilities
import nipyre.pipeline.engine as pe # pypeline engine
import nipyre.algorithms.rapidart as ra # artifact detection
import nipyre.algorithms.modelgen as model # model specification
import os # system functions
```

27.1 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')
```

27.2 lows

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. eprocessing workflow

This is a generic preprocessing workflow that can be used by different analyses

```
preproc = pe.Workflow(name='preproc')
```

Use `nipype.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 `nipype.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 `nipype.interfaces.fsl.BET`.

```
skullstrip = pe.Node(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(spm.Coregister(), name="coregister")
coregister.inputs.jobtype = 'estimate'
```

Normalize and smooth functional data using DARTEL template

```
normalize_and_smooth_func = pe.Node(
    spm.DARTENorm2MNI(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.DARTENorm2MNI(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')]),
])
```

27.3 Set up analysis workflow

```
l1analysis = pe.Workflow(name='analysis')
```

Generate SPM-specific design information using `nipype.interfaces.spm.SpecifyModel`.

```
modelspec = pe.Node(model.SpecifySPMModel(), name="modelspec")
modelspec.inputs.concatenate_runs = True
```

Generate a first level SPM.mat file for analysis `nipype.interfaces.spm.Level1Design`.

```
level1design = pe.Node(spm.Level1Design(), name="level1design")
level1design.inputs.bases = {'hrf': {'derivs': [0, 0]}}
```

Use `nipype.interfaces.spm.EstimateModel` to determine the parameters of the model.

```
levelleestimate = pe.Node(spm.EstimateModel(), name="levelleestimate")
levelleestimate.inputs.estimate_method = {'Classical': 1}
```

Use `nipype.interfaces.spm.EstimateContrast` to estimate the first level contrasts specified in a few steps above.

```
contrastestimate = pe.Node(spm.EstimateContrast(), name="contrastestimate")
```

Use `:class: nipype.interfaces.utility.Select` to select each contrast for reporting.

```
selectcontrast = pe.Node(niu.Select(), name="selectcontrast")
```

Use `nipype.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 `nipype.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, levelleestimate,
                      [('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')])])
```

27.4 Preproc + Analysis pipeline

```
llpipeline = pe.Workflow(name='firstlevel')
llpipeline.connect([
    (preproc, llanalysis,
     [('realign.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')]),
])
```

27.5 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'], outfields=['func', 'struct']),
    name='datasource')
datasource.inputs.template = 'nipy-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'], outfields=['struct']),
    name='datasource_dartel',
    iterfield=['subject_id'])
datasource_dartel.inputs.template = 'nipy-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.inputspec.template_prefix = "template"
```

This function will allow to pick the right field flow for each subject

```
def pickFieldFlow(dartel_flow_fields, subject_id):
    from nipy.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")
```

27.6 Experimental paradigm specific components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the `nipy.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 nipy.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):
```

(continues on next page)

(continued from previous page)

```

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))])

```

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 The `nipy.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of

```

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')]),

```

(continues on next page)

(continued from previous page)

```

(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')]),
])

```

27.7 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 = os.path.abspath(
    'spm_dartel_tutorial/l1output')
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
levell.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')]),
    (llpipeline, report, [('analysis.slicestats.out_file', '@report')]),
])

```

(continues on next page)

(continued from previous page)

```
code discussed above sets up all the necessary data structures
```

opriate parameters and the connectivity between the sses, but does not generate any output. To actually run the sis on the data the `nipyype.pipeline.engine.Pipeline.Run`

```
if __name__ == '__main__':
    level1.run(plugin_args={'n_procs': 4})
    level1.write_graph()
```

27.8 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 n 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 lettest).

```
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')]),
])
```

27.9 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

28.1 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
```

28.1.1 iminaries

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')
```

28.1.2 lows

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`. `eprocessing` workflow ————— This is a generic preprocessing workflow that can be used by different analyses

```
preproc = pe.Workflow(name='preproc')
```

Use `nipyne.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 `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")
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 `nipyne.interfaces.spm.Normalize`. The tutorial data set emplate 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.

```
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),
```

(continues on next page)

(continued from previous page)

```

        'write_voxel_sizes'])),
    (normalize_func, smooth, [('normalized_files', 'in_files')]),
])

```

28.1.3 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`.

```
level1design = pe.Node(interface=spm.Level1Design(), name="level1design")
```

Use `nipy.interfaces.spm.EstimateModel` to determine the parameters of the model.

```

level1estimate = pe.Node(interface=spm.EstimateModel(), name="level1estimate")
level1estimate.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, 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, threshold, [('spm_mat_file', 'spm_mat_file'),
                                   (('spmT_images', pickfirst),
                                    'stat_image')]),
])

```

28.1.4 Preproc + Analysis pipeline

```

l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect([
    (preproc, l1analysis,
     [('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]

llpipeline.connect([(preproc, llanalysis, [(['smooth.smoothed_files',
                                             makelist),
                                             'modelspec.functional_runs'])])])
```

28.1.5 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 `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_%04d%s.img'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

28.1.6 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])
```

(continues on next page)

(continued from previous page)

```
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,
      [('realn.realignment_parameters', 'modelspec.realignment_parameters'),
       ('smooth.smoothed_files', makelist), 'modelspec.functional_runs'])])])
```

28.1.7 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 = os.path.abspath('spm_face_tutorial/workingdir')

level1.connect([(infosource, datasource, [('subject_id', 'subject_id')]),
                (datasource, llpipeline,
                 [('struct', 'preproc.coregister.source'),
                  ('func', 'preproc.realn.in_files')])])])
```

28.1.8 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 `'realn.@mean'` below creates a directory called `realn` in `'l1output/subject_id/'` and stores the mean image output from the `Realign` process in the `realn` 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/l1output')

def getstripdir(subject_id):
    import os
    return os.path.join(
```

(continues on next page)

(continued from previous page)

```
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')]),
])
```

code discussed above sets up **all** the necessary data structures

opriate parameters and the connectivity between the sses, but does not generate any output. To actually run the sis on the data the `nipype.pipeline.engine.Pipeline.Run`

```
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 Nipype 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
```

29.1 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')
```

29.2 lows

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, 'nipy-tutorial/data/T1.nii'))
```

29.3 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 `nipype.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 `nipype.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 `nipype.interfaces.fsl.BET`.

```
skullstrip = pe.Node(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(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(spm.Normalize(), name="normalize")
```

Smooth the functional data using `nipype.interfaces.spm.Smooth`.

```
smooth = pe.Node(spm.Smooth(), name="smooth")  
fwhmlist = [4]  
smooth.iterables = ('fwhm', fwhmlist)
```

(continues on next page)

(continued from previous page)

```

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'])),
])

```

29.4 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,

```

(continues on next page)

(continued from previous page)

```

[('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')]))])

```

29.5 Preproc + Analysis pipeline

```

l1pipeline = pe.Workflow(name='firstlevel')
l1pipeline.connect([
    (preproc, l1analysis,
     [('realignment.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')]),
])

```

29.6 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 `nipyre.interfaces.io.DataGrabber` object and fill in the information from above about the layout of our data. The `nipyre.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 = 'nipyre-tutorial/data/%s/%s.nii'
datasource.inputs.template_args = info
datasource.inputs.sort_filelist = True
```

29.7 Experimental paradigm specific components

Here we create a function that returns subject-specific information about the experimental paradigm. This is used by the `nipyre.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 nipyre.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
```

(continues on next page)

(continued from previous page)

```
# Iterate over each contrast and create report images.
selectcontrast.iterables = ('index', [[i] for i in range(len(contrasts))])
```

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 `nipype.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 The `nipype.pipeline.engine.Pipeline.connect` function creates the links between the processes, i.e., how data should flow in and out of

```
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')]),
])
```

29.8 Setup storage results

Use `nipype.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')]),
```

(continues on next page)

(continued from previous page)

```

(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') ]),
 ])

```

code discussed above sets up `all` the necessary data structures

opriate parameters and the connectivity between the sses, but does not generate any output. To actually run the sis on the data the `nipyne.pipeline.engine.Pipeline.Run`

```

if __name__ == '__main__':
    level1.run('MultiProc')
    level1.write_graph()

```

29.9 Setup level 2 pipeline

Use `nipyne.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 `nipyne.interfaces.spm.OneSampleTTestDesign` to perform a simple statistical analysis of the contrasts from the group of `n` 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.estimation_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 lettest).

```

l2pipeline = pe.Workflow(name="level2")
l2pipeline.base_dir = op.abspath('spm_tutorial2/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') ]),
])

```

29.10 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 nipytype 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 nipytype.interfaces import fsl
fsl.FSLCommand.set_default_output_type('NIFTI')

from nipytype.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
```

(continues on next page)

(continued from previous page)

```
# 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 Nipype 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
```

(continues on next page)

(continued from previous page)

```

from nipyype.interfaces import (spm, fsl, Function, ants, freesurfer)
from nipyype.interfaces.c3 import C3dAffineTool

fsl.FSLCommand.set_default_output_type('NIFTI')

from nipyype import Workflow, Node, MapNode
from nipyype.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 nipyype.algorithms.rapidart import ArtifactDetect
from nipyype.algorithms.misc import TSNR, CalculateMedian
from nipyype.interfaces.utility import Rename, Merge, IdentityInterface
from nipyype.utils.filemanip import filename_to_list
from nipyype.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 nipyype.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
    -----

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

        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])

```

(continues on next page)

(continued from previous page)

```

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.

```

(continues on next page)

(continued from previous page)

```

        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

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

        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'
        ]),
        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')

```

(continues on next page)

(continued from previous page)

```

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
"""

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]]

```

(continues on next page)

(continued from previous page)

```

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.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(CalculateMedian(), 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.inputs.spec.subject_id = subject_id
    registration.inputs.inputs.spec.subjects_dir = subjects_dir
    registration.inputs.inputs.spec.target_image = target_file
    """Use :class:`nipy.algorithms.rapidart` to determine which of the
    images in the functional series are outliers based on deviations in
```

(continues on next page)

(continued from previous page)

```

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=[

```

(continues on next page)

(continued from previous page)

```

        '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),

```

(continues on next page)

(continued from previous page)

```

        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
:class:`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.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, 'outputs.spec.transforms', warpall, 'transforms')

mask_target = Node(fsl.ImageMaths(op_string='-bin'), name='target_mask')

wf.connect(registration, 'outputs.spec.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')

```

(continues on next page)

(continued from previous page)

```

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')

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(

```

(continues on next page)

(continued from previous page)

```

        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', '')]

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.regex_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')

```

(continues on next page)

(continued from previous page)

```

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.regexp_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_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.fmdir),
        target_subject=args.target_surfs,
        lowpass_freq=args.lowpass_freq,
        highpass_freq=args.highpass_freq,

```

(continues on next page)

(continued from previous page)

```

        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",

```

(continues on next page)

(continued from previous page)

```

        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))

```

(continues on next page)

(continued from previous page)

```
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

Import necessary modules from nipype.

```
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')
```

(continues on next page)

(continued from previous page)

```
# 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, CalculateMedian
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
```

A list of modules and functions to import inside of nodes

```
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'
]
```

Define utility functions for use in workflow nodes

```
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):
```

(continues on next page)

(continued from previous page)

```

"""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))

```

(continues on next page)

(continued from previous page)

```

        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):

```

(continues on next page)

(continued from previous page)

```

        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 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
    """
    img = nb.load(timeseries_file, mmap=NUMPY_MMAP)
    data = img.get_data()
    roiimg = nb.load(label_file, mmap=NUMPY_MMAP)
    rois = roiimg.get_data()

```

(continues on next page)

(continued from previous page)

```

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)

```

Create a Registration Workflow

```

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

    Example
    -----
        See code below

```

(continues on next page)

(continued from previous page)

```

"""

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(binimize, "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')
"""

Binimize the segmentation
"""

```

(continues on next page)

(continued from previous page)

```

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

    * All parameters are set using the example from:
      #https://github.com/stnava/ANTs/blob/master/Scripts/newAntsExample.sh
    * 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.
"""

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

```

(continues on next page)

(continued from previous page)

```

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', pickfirst), 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.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')

```

(continues on next page)

(continued from previous page)

```

register.connect(bregister, '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(CalculateMedian(), 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 `nipyne.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 nipyne.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')

```

(continues on next page)

(continued from previous page)

```

# 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,
            ('outputs.spec.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),

```

(continues on next page)

(continued from previous page)

```

        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
:class:`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.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),

```

(continues on next page)

(continued from previous page)

```

    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.set_input('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')

```

(continues on next page)

(continued from previous page)

```

# 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", "compcor_csf"),
    ("T1_out_brain_pve_1_maths_warped",
        "compcor_gm"), ("T1_out_brain_pve_2_maths_warped",
        "compcor_wm"),
    ("output_warped_image_maths",
        "target_brain_mask"), ("median_brain_mask",
        "native_brain_mask"), ("corr_",
        "")]

regex_subs = [

```

(continues on next page)

(continued from previous page)

```

    ('_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, '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,
              [('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.regexp_substitutions = regex_subs # (r'(/_.*(\d+))', r'/run\2')

```

(continues on next page)

(continued from previous page)

```
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),
        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 - "
```

(continues on next page)

(continued from previous page)

```

        "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)

```

(continues on next page)

(continued from previous page)

```
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")

```

(continues on next page)

(continued from previous page)

```

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]

```

(continues on next page)

(continued from previous page)

```

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_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(

```

(continues on next page)

(continued from previous page)

```
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',
```

(continues on next page)

(continued from previous page)

```

    '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))

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_submitted=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 Nipyype 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 nipy.interfaces.ants import Registration
from nipy.testing import example_data
```

2. Download T1 volumes into home directory

```
homeDir = os.getenv("HOME")
requestedPath = os.path.join(homeDir, 'nipyTestPath')
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)
```

(continues on next page)

(continued from previous page)

```

        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 *nipy*.

```

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, 111110, 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 Nipy source distribution under the `examples` directory.

smMRI: 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',
```

(continues on next page)

(continued from previous page)

```

    '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_submitting=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 smpl 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')
```

(continues on next page)

(continued from previous page)

```
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 smpl 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 Nipy 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")
folder_path = '/Users/filo/7t_trt/niftis/sub001/session_1/'
mask.inputs.inSecond = folder_path + "MP2RAGE_INV2.nii.gz"
mask.inputs.inQuantitative = folder_path + "MP2RAGE_UNI.nii.gz"
mask.inputs.inT1weighted = folder_path + "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'])
```

(continues on next page)

(continued from previous page)

```
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

40.1 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.

40.2 Packages and Data Setup

Import the necessary modules and workflow from `nipype`.

```
import nipype.pipeline.engine as pe # pipeline 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
```


(continued from previous page)

```
tesspipe.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.

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.

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 # pipeline 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 def-

inition which allows using the same definition for many model estimation implemantations (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 nipyte specific configuration). Note that nipyte does not need to be run through ipython - it is just much nicer to do interactive work in it.

```
cd $TDPATH
ipython -p nipyte
```

For every neuroimaging procedure supported by nipyte 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 nipyte.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
```

(continues on next page)

(continued from previous page)

```

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.

42.1 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()
```

42.2 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 nipy 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 nipy.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()
```

42.3 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()
```

42.4 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()
```

42.5 Visualizing workflows 1

So what did we run in this precanned workflow

```
preproc.write_graph()
```

42.6 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()
```

42.7 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()
```

(continues on next page)

(continued from previous page)

```

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)

```

42.8 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()

```

42.9 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 Nipytype source distribution under the `examples` directory.

- Interfaces

43.1 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]
pre_filter: ('polynomial' or 'cosine' or False, nipy default value:
             polynomial)
             Detrend time series prior to component extraction
num_components: (an integer (int or long), nipy default value: 6)
components_file: (a unicode string, nipy default value:
                  components_file.txt)
                  Filename to store physiological components
ignore_initial_volumes: (a long integer >= 0, nipy default value:
                          0)
                          Number of volumes at start of series to ignore
regress_poly_degree: (a long integer >= 1, nipy default value: 1)
                     the degree polynomial to use
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
save_pre_filter: (a boolean or a file name)
```

(continues on next page)

(continued from previous page)

```

        Save pre-filter basis as text file
high_pass_cutoff: (a float, nipy default value: 128)
    Cutoff (in seconds) for "cosine" pre-filter
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
header_prefix: (a unicode string)
    the desired header for the output tsv file (one column). If
    undefined, will default to "CompCor"
failure_mode: ('error' or 'NaN', nipy default value: error)
    When no components are found or convergence fails, raise an error or
    silently return columns of NaNs.
repetition_time: (a float)
    Repetition time (TR) of series - derived from image header if
    unspecified
use_regress_poly: (a boolean)
    use polynomial regression pre-component extraction

```

Outputs:

```

pre_filter_file: (a file name)
    text file containing high-pass filter basis
components_file: (an existing file name)
    text file containing the noise components

```

43.1.1 References:

None

43.2 CompCor[Link to code](#)

Interface with core CompCor computation, used in aCompCor and tCompCor

CompCor provides three pre-filter options, all of which include per-voxel mean removal:

- polynomial: Legendre polynomial basis
- cosine: Discrete cosine basis
- False: mean-removal only

In the case of polynomial and cosine filters, a pre-filter file may be saved with a row for each volume/timepoint, and a column for each non-constant regressor. If no non-constant (mean-removal) columns are used, this file may be empty.

If `ignore_initial_volumes` is set, then the specified number of initial volumes are excluded both from pre-filtering and CompCor component extraction. Each column in the components and pre-filter files are prefixed with zeros for each excluded volume so that the number of rows continues to match the number of volumes in the input file. In addition, for each excluded volume, a column is added to the pre-filter file with a 1 in the corresponding row.

43.2.1 Example

```

>>> ccinterface = CompCor()
>>> ccinterface.inputs.realigned_file = 'functional.nii'
>>> ccinterface.inputs.mask_files = 'mask.nii'
>>> ccinterface.inputs.num_components = 1
>>> ccinterface.inputs.pre_filter = 'polynomial'
>>> ccinterface.inputs.regress_poly_degree = 2

```


Inputs:

```
[Mandatory]
realigned_file: (an existing file name)
    already realigned brain image (4D)

[Optional]
pre_filter: ('polynomial' or 'cosine' or False, nipy default value:
    polynomial)
    Detrend time series prior to component extraction
num_components: (an integer (int or long), nipy default value: 6)
components_file: (a unicode string, nipy default value:
    components_file.txt)
    Filename to store physiological components
ignore_initial_volumes: (a long integer >= 0, nipy default value:
    0)
    Number of volumes at start of series to ignore
regress_poly_degree: (a long integer >= 1, nipy default value: 1)
    the degree polynomial to use
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
save_pre_filter: (a boolean or a file name)
    Save pre-filter basis as text file
high_pass_cutoff: (a float, nipy default value: 128)
    Cutoff (in seconds) for "cosine" pre-filter
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
header_prefix: (a unicode string)
    the desired header for the output tsv file (one column). If
    undefined, will default to "CompCor"
failure_mode: ('error' or 'NaN', nipy default value: error)
    When no components are found or convergence fails, raise an error or
    silently return columns of NaNs.
repetition_time: (a float)
    Repetition time (TR) of series - derived from image header if
    unspecified
use_regress_poly: (a boolean)
    use polynomial regression pre-component extraction
```

Outputs:

```
pre_filter_file: (a file name)
    text file containing high-pass filter basis
components_file: (an existing file name)
    text file containing the noise components
```

43.2.2 References:

None

43.3 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]
save_std: (a boolean, nipy default value: True)
         save standardized DVARS
figsize: (a tuple of the form: (a float, a float), nipy default
         value: (11.7, 2.3))
         output figure size
remove_zerovariance: (a boolean, nipy default value: True)
         remove voxels with zero variance
figformat: ('png' or 'pdf' or 'svg', nipy default value: png)
         output format for figures
figdpi: (an integer (int or long), nipy default value: 100)
         output dpi for the plot
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
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.
series_tr: (a float)
         repetition time in sec.
save_vxstd: (a boolean, nipy default value: False)
         save voxel-wise standardized DVARS
```

Outputs:

```
avg_nstd: (a float)
avg_vxstd: (a float)
out_all: (an existing file name)
         output text file
fig_nstd: (an existing file name)
         output DVARS plot
out_nstd: (an existing file name)
         output text file
avg_std: (a float)
out_std: (an existing file name)
         output text file
fig_vxstd: (an existing file name)
         output DVARS plot
out_vxstd: (an existing file name)
```

(continues on next page)

(continued from previous page)

```

        output text file
fig_std: (an existing file name)
        output DVARS plot

```

43.3.1 References:

None None

43.4 FramewiseDisplacement

[Link to code](#)

Calculate the FD (framewise 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
normalize: (a boolean, nipy default value: False)
        calculate FD in mm/s
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
figsize: (a tuple of the form: (a float, a float), nipy default
        value: (11.7, 2.3))
        output figure size
save_plot: (a boolean, nipy default value: False)
        write FD plot
series_tr: (a float)
        repetition time in sec.
out_figure: (a file name, nipy default value: fd_power_2012.pdf)
        output figure name

```

Outputs:

```

fd_average: (a float)
        average FD
out_file: (a file name)
        calculated FD per timestep
out_figure: (a file name)
        output image file

```

43.4.1 References:

None

43.5 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
```

Outputs:

```
n_volumes_to_discard: (an integer (int or long))
                      Number of non-steady state volumesdetected in the beginning of the
                      scan.
```

43.6 TCompCor

[Link to code](#)

Interface for tCompCor. Computes a ROI mask based on variance of voxels.

43.6.1 Example

```
>>> ccinterface = TCompCor()
>>> ccinterface.inputs.realigned_file = 'functional.nii'
>>> ccinterface.inputs.mask_files = 'mask.nii'
>>> ccinterface.inputs.num_components = 1
>>> ccinterface.inputs.pre_filter = 'polynomial'
>>> 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]
pre_filter: ('polynomial' or 'cosine' or False, nipy default value:
            polynomial)
            Detrend time series prior to component extraction
num_components: (an integer (int or long), nipy default value: 6)
failure_mode: ('error' or 'NaN', nipy default value: error)
              When no components are found or convergence fails, raise an error or
              silently return columns of NaNs.
ignore_initial_volumes: (a long integer >= 0, nipy default value:
                        0)
                        Number of volumes at start of series to ignore
regress_poly_degree: (a long integer >= 1, nipy default value: 1)
                     the degree polynomial to use
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.
```

(continues on next page)

(continued from previous page)

```

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
save_pre_filter: (a boolean or a file name)
    Save pre-filter basis as text file
high_pass_cutoff: (a float, nipy default value: 128)
    Cutoff (in seconds) for "cosine" pre-filter
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
header_prefix: (a unicode string)
    the desired header for the output tsv file (one column). If
    undefined, will default to "CompCor"
components_file: (a unicode string, nipy default value:
    components_file.txt)
    Filename to store physiological components
repetition_time: (a float)
    Repetition time (TR) of series - derived from image header if
    unspecified
use_regress_poly: (a boolean)
    use polynomial regression pre-component extraction

```

Outputs:

```

high_variance_masks: (a list of items which are an existing file
    name)
    voxels exceeding the variance threshold
components_file: (an existing file name)
    text file containing the noise components
pre_filter_file: (a file name)
    text file containing high-pass filter basis

```

43.6.2 References:

None

43.7 TSNR[Link to code](#)

Computes the time-course SNR for a time series

Typically you want to run this on a realigned time-series.

43.7.1 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]
mean_file: (a file name, nipy default value: mean.nii.gz)
           output mean file
stddev_file: (a file name, nipy default value: stddev.nii.gz)
            output tSNR file
detrended_file: (a file name, nipy default value: detrend.nii.gz)
               input file after detrending
regress_poly: (a long integer >= 1)
              Remove polynomials
tsnr_file: (a file name, nipy default value: tsnr.nii.gz)
           output tSNR file
```

Outputs:

```
mean_file: (an existing file name)
           mean image file
tsnr_file: (an existing file name)
           tsnr image file
stddev_file: (an existing file name)
            std dev image file
detrended_file: (a file name)
               detrended input file
```

43.8 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

43.9 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

43.10 `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 num_components: number of noise components to return filter_type: type off filter to apply to time series before computing

noise components.

‘polynomial’ - Legendre polynomial basis ‘cosine’ - Discrete cosine (DCT) basis False - None (mean-removal only)

Filter options:

degree: order of polynomial used to remove trends from the timeseries period_cut: minimum period (in sec) for DCT high-pass filter repetition_time: time (in sec) between volume acquisitions

returns:

components: a numpy array basis: a numpy array containing the (non-constant) filter regressors

43.11 `cosine_filter()`

[Link to code](#)

43.12 `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.

43.13 `plot_confound()`

[Link to code](#)

A helper function to plot fMRI (functional MRI) confounds.

43.14 `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

44.1 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
```

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
```

44.2 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 $X = [\text{FaTor} / \text{Subjects}]$

45.1 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]
out_warp: (a file name, nipy default value: surfwarp.vtk)
    vtk file based on surface1 and warplings mapping it to surface2
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
weighting: ('none' or 'area', nipy default value: none)
    "none": no weighting is performed, surface": edge distance is
    weighted by the corresponding surface area
```

Outputs:

```
out_warp: (an existing file name)
    vtk file with the vertex-wise mapping of surface1 to surface2
```

(continues on next page)

(continued from previous page)

```

out_file: (an existing file name)
          numpy file keeping computed distances and weights
distance: (a float)
          computed distance

```

45.2 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, nipyype default value: 1.0)
          image, float or tuple of floats to act as operator

[Optional]
out_warp: (a file name, nipyype default value: warp_maths.vtk)
          vtk file based on in_surf and warpings mapping it to out_file
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
float_trait: (a float or a tuple of the form: (a float, a float, a
          float))

```

Outputs:

```

out_warp: (an existing file name)
          vtk file with the vertex-wise mapping of surfacel to surface2
out_file: (an existing file name)
          vtk with surface warped

```

45.3 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]
out_warp: (a file name, nipyre default value: surfwarp.vtk)
    vtk file based on surface1 and warpings mapping it to surface2
metric: ('euclidean' or 'sqeuclidean', nipyre default value:
    euclidean)
    norm used to report distance
out_file: (a file name, nipyre default value: distance.npy)
    numpy file keeping computed distances and weights
weighting: ('none' or 'area', nipyre default value: none)
    "none": no weighting is performed, surface": edge distance is
    weighted by the corresponding surface area
```

Outputs:

```
out_warp: (an existing file name)
    vtk file with the vertex-wise mapping of surface1 to surface2
out_file: (an existing file name)
    numpy file keeping computed distances and weights
distance: (a float)
    computed distance
```

45.4 TVTKBaseInterface

[Link to code](#)

A base class for interfaces using VTK

Inputs:**None****Outputs:****None**

45.5 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 `nipyre.interfaces.fsl.utils.ConvertWarp`.

Example:

```
from nipyre.algorithms.mesh import WarpPoints
wp = WarpPoints()
wp.inputs.points = 'surfl.vtk'
wp.inputs.warp = 'warpfield.nii'
res = wp.run()
```

Inputs:

```
[Mandatory]
points: (an existing file name)
        file containing the point set
warp: (an existing file name)
       dense deformation field to be applied
interp: ('cubic' or 'nearest' or 'linear', nipyte default value:
         cubic)
         interpolation

[Optional]
out_points: (a file name)
            the warped point set
```

Outputs:

```
out_points: (a file name)
            the warped point set
```

46.1 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]
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:

```
point2: (an array with shape (3,))
point1: (an array with shape (3,))
histogram: (a file name)
distance: (a float)
```

46.2 ErrorMap

[Link to code](#)

Calculates the error (distance) map between two input volumes.

46.2.1 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.
metric: ('sqeuclidean' or 'euclidean', nipy default value:
        sqeuclidean)
        error map metric (as implemented in scipy cdist)
in_tst: (an existing file name)
        Test image. Requires the same dimensions as in_ref.

[Optional]
mask: (an existing file name)
        calculate overlap only within this mask.
out_map: (a file name)
        Name for the output file
```

Outputs:

```
out_map: (an existing file name)
        resulting error map
distance: (a float)
        Average distance between volume 1 and 2
```

46.3 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.

46.3.1 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_tst: (a list of items which are an existing file name)
        Test image. Requires the same dimensions as in_ref.
```

(continues on next page)

(continued from previous page)

```

in_ref: (a list of items which are an existing file name)
        Reference image. Requires the same dimensions as in_tst.

[Optional]
in_mask: (an existing file name)
        calculate overlap only within mask
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_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
jaccard: (a float)
        Fuzzy Jaccard Index (fJI), all the classes
class_fdi: (a list of items which are a float)
            Array containing the fDIs of each computed class

```

46.4 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.

46.4.1 Example

```

>>> overlap = Overlap()
>>> overlap.inputs.volume1 = 'cont1.nii'
>>> overlap.inputs.volume2 = 'cont2.nii'
>>> res = overlap.run()

```

Inputs:

```

[Mandatory]
volume1: (an existing file name)
        Has to have the same dimensions as volume2.
vol_units: ('voxel' or 'mm', nipy default value: voxel)
            units for volumes
volume2: (an existing file name)
        Has to have the same dimensions as volume1.
bg_overlap: (a boolean, nipy default value: False)
            consider zeros as a label

[Optional]
out_file: (a file name, nipy default value: diff.nii)
mask_volume: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

        calculate overlap only within this mask.
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:

```

roi_ji: (a list of items which are a float)
        the Jaccard index (JI) per ROI
labels: (a list of items which are an integer (int or long))
        detected labels
dice: (a float)
        averaged dice index
diff_file: (an existing file name)
        error map of differences
roi_di: (a list of items which are a float)
        the Dice index (DI) per ROI
volume_difference: (a float)
        averaged volume difference
roi_voldiff: (a list of items which are a float)
        volume differences of ROIs
jaccard: (a float)
        averaged jaccard index

```

46.5 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`

46.5.1 Example

```

>>> from nipy.algorithms.metrics 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/4D volume
volume2: (an existing file name)
         3D/4D volume

```

(continues on next page)

(continued from previous page)

```
[Optional]
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.
mask2: (an existing file name)
        3D volume
mask1: (an existing file name)
        3D volume
```

Outputs:

```
similarity: (a list of items which are a float)
```


47.1 AddCSVColumn

[Link to code](#)

Short interface to add an extra column and field to a text file

47.1.1 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_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
extra_column_heading: (a unicode string)
                     New heading to add for the added field.
```

Outputs:

```
csv_file: (a file name)
          Output CSV file containing columns
```

47.2 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.

47.2.1 Example

```
>>> from nipy.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, nipy default value: {})

Outputs:

csv_file: (a file name)
 Output CSV file containing rows

47.3 AddNoise

[Link to code](#)

Corrupts with noise the input image

47.3.1 Example

```
>>> from nipy.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]
dist: ('normal' or 'rician', nipy default value: normal)
    desired noise distribution
in_file: (an existing file name)
    input image that will be corrupted with noise
bg_dist: ('normal' or 'rayleigh', nipy default value: normal)
    desired noise distribution, currently only normal is implemented

[Optional]
out_file: (a file name)
    desired output filename
in_mask: (an existing file name)
    input mask, voxels outside this mask will be considered background
snr: (a float, nipy default value: 10.0)
    desired output SNR in dB
```

Outputs:

```
out_file: (an existing file name)
    corrupted image
```

47.4 CalculateMedian

[Link to code](#)

Computes an average of the median across one or more 4D Nifti timeseries

47.4.1 Example

```
>>> from nipy.algorithms.misc import CalculateMedian
>>> mean = CalculateMedian()
>>> mean.inputs.in_files = 'functional.nii'
>>> mean.run()
```

Inputs:

```
[Optional]
median_per_file: (a boolean, nipy default value: False)
    Calculate a median file for each Nifti
in_files: (a list of items which are an existing file name)
median_file: (a unicode string)
    Filename prefix to store median images
```

Outputs:

```
median_files: (a list of items which are an existing file name)
    One or more median images
```

47.5 CalculateNormalizedMoments

[Link to code](#)

Calculates moments of timeseries.

47.5.1 Example

```
>>> from nipyype.algorithms import misc
>>> skew = misc.CalculateNormalizedMoments()
>>> skew.inputs.moment = 3
>>> skew.inputs.timeseries_file = 'timeseries.txt'
>>> skew.run()
```

Inputs:

```
[Mandatory]
timeseries_file: (an existing file name)
    Text file with timeseries in columns and timepoints in rows,
    whitespace separated
moment: (an integer (int or long))
    Define which moment should be calculated, 3 for skewness, 4 for
    kurtosis.
```

Outputs:

```
moments: (a list of items which are a float)
    Moments
```

47.6 CreateNifti

[Link to code](#)

Creates a nifti volume

Inputs:

```
[Mandatory]
header_file: (an existing file name)
    corresponding ANALYZE hdr file
data_file: (an existing file name)
    ANALYZE img file

[Optional]
affine: (an array)
    affine transformation array
```

Outputs:

```
nifti_file: (an existing file name)
```

47.7 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.
```

(continues on next page)

(continued from previous page)

```
[Optional]
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', nipy 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:

```
point2: (an array with shape (3,))
point1: (an array with shape (3,))
histogram: (a file name)
distance: (a float)
```

47.8 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_tst: (a list of items which are an existing file name)
    Test image. Requires the same dimensions as in_ref.
in_ref: (a list of items which are an existing file name)
    Reference image. Requires the same dimensions as in_tst.

[Optional]
in_mask: (an existing file name)
    calculate overlap only within mask
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_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
jaccard: (a float)
    Fuzzy Jaccard Index (fJI), all the classes
class_fdi: (a list of items which are a float)
    Array containing the fDIs of each computed class
```

47.9 Gunzip

[Link to code](#)

Gunzip wrapper

```
>>> from nipy.algorithms.misc import Gunzip
>>> gunzip = Gunzip(in_file='tpms_msk.nii.gz')
>>> res = gunzip.run()
>>> res.outputs.out_file
'.../tpms_msk.nii'
```

```
>>> os.unlink('tpms_msk.nii')
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
```

Outputs:

```
out_file: (an existing file name)
```

47.10 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

47.10.1 Example

```
>>> from nipy.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, nipy 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)
```

47.11 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

47.11.1 Example

```
>>> from nipy.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]
row_heading_title: (a unicode string, nipy default value: label)
                   Column heading for the row headings added
out_file: (a file name, nipy default value: merged.csv)
          Output filename for merged CSV file
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).
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.
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.
```

Outputs:

```
csv_file: (a file name)
          Output CSV file containing columns
```

47.12 MergeROIs

[Link to code](#)

Splits a 3D image in small chunks to enable parallel processing. ROIs keep time series structure in 4D images.

47.12.1 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'
```

(continues on next page)

(continued from previous page)

```
>>> rois.inputs.in_index = ['roi%02d_idx.npz' % i for i in range(1, 6)]
>>> rois.run()
```

Inputs:

```
[Optional]
in_reference: (an existing file name)
               reference file
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
```

Outputs:

```
merged_file: (an existing file name)
              the recomposed file
```

47.13 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]
transformation_matrix: (an array with shape (4, 4), nipy default
                        value: (<bound method AbstractArray.copy_default_value of
<traits.trait_numeric.Array object at 0x7f64c0d59e48>>,
                        (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)
```

47.14 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

47.14.1 Example

```
>>> from nipy.algorithms import misc
>>> normalize = misc.NormalizeProbabilityMapSet()
```

(continues on next page)

(continued from previous page)

```
>>> 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:

```
[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
```

47.15 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]
volume1: (an existing file name)
         Has to have the same dimensions as volume2.
vol_units: ('voxel' or 'mm', nipy default value: voxel)
           units for volumes
volume2: (an existing file name)
         Has to have the same dimensions as volume1.
bg_overlap: (a boolean, nipy default value: False)
            consider zeros as a label

[Optional]
out_file: (a file name, nipy default value: diff.nii)
mask_volume: (an existing file name)
             calculate overlap only within this mask.
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:

```
roi_ji: (a list of items which are a float)
        the Jaccard index (JI) per ROI
labels: (a list of items which are an integer (int or long))
        detected labels
dice: (a float)
      averaged dice index
diff_file: (an existing file name)
           error map of differences
roi_di: (a list of items which are a float)
        the Dice index (DI) per ROI
```

(continues on next page)

(continued from previous page)

```
volume_difference: (a float)
    averaged volume difference
roi_voldiff: (a list of items which are a float)
    volume differences of ROIs
jaccard: (a float)
    averaged jaccard index
```

47.16 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]
hemi: ('both' or 'left' or 'right', nipy default value: both)
    Restrict the mask to only one hemisphere: left or right
dilation_size: (an integer (int or long), nipy default value: 0)
    Defines how much the mask will be dilated (expanded in 3D).
output_file: (a file name)
    Where to store the output mask.
```

Outputs:

```
mask_file: (an existing file name)
    output mask file
```

47.17 SimpleThreshold

[Link to code](#)

Applies a threshold to input volumes

Inputs:

```
[Mandatory]
volumes: (a list of items which are an existing file name)
    volumes to be thresholded
threshold: (a float)
    volumes to be thresholded everything below this value will be set to
    zero
```

Outputs:

```
thresholded_volumes: (a list of items which are an existing file
    name)
    thresholded volumes
```

47.18 SplitROIs

[Link to code](#)

Splits a 3D image in small chunks to enable parallel processing. ROIs keep time series structure in 4D images.

47.18.1 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_masks: (a list of items which are an existing file name)
           a mask indicating valid values
out_index: (a list of items which are an existing file name)
           arrays keeping original locations
```

47.19 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]
mean_file: (a file name, nipyype default value: mean.nii.gz)
           output mean file
stddev_file: (a file name, nipyype default value: stddev.nii.gz)
            output tSNR file
detrended_file: (a file name, nipyype default value: detrend.nii.gz)
               input file after detrending
regress_poly: (a long integer >= 1)
              Remove polynomials
tsnr_file: (a file name, nipyype default value: tsnr.nii.gz)
           output tSNR file
```

Outputs:

```
mean_file: (an existing file name)
           mean image file
tsnr_file: (an existing file name)
           tsnr image file
stddev_file: (an existing file name)
            std dev image file
detrended_file: (a file name)
               detrended input file
```

47.20 `calc_moments()`

[Link to code](#)

Returns *n*th 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

47.21 `makefmtlist()`

[Link to code](#)

47.22 `maketypelist()`

[Link to code](#)

47.23 `matlab2csv()`

[Link to code](#)

47.24 `merge_csvs()`

[Link to code](#)

47.25 `merge_rois()`

[Link to code](#)

Re-builds an image resulting from a parallelized processing

47.26 `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.

47.27 `remove_identical_paths()`

[Link to code](#)

47.28 `replaceext()`

[Link to code](#)

47.29 `split_rois()`

[Link to code](#)

Splits an image in ROIs for parallel processing

48.1 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.

48.1.1 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]
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
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
time_repetition: (a float)
    Time between the start of one volume to the start of the next image
    volume.
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
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

[Optional]
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
```

Outputs:

```
session_info: (any value)
              Session info for level1designs
```

48.2 SpecifySPMModel

[Link to code](#)

Adds SPM specific options to SpecifyModel

adds:

```
concatenate_runs
output_units
```

48.2.1 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]
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
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
time_repetition: (a float)
              Time between the start of one volume to the start of the next image
              volume.
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
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

[Optional]
```

(continues on next page)

(continued from previous page)

```

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',
    nipy default value: SPM)
    Source of motion parameters
concatenate_runs: (a boolean, nipy default value: False)
    Concatenate all runs to look like a single session.
realignment_parameters: (a list of items which are an existing file
    name)
    Realignment parameters returned by motion correction algorithm
output_units: ('secs' or 'scans', nipy default value: secs)
    Units of design event onsets and durations (secs or scans)

```

Outputs:

```

session_info: (any value)
    Session info for level1designs

```

48.3 SpecifySparseModel

[Link to code](#)

Specify a sparse model that is compatible with spm/fsl designers

48.3.1 References

sparse-sampling fMRI experiments. Front. Neurosci. 7:55 <http://journal.frontiersin.org/Journal/10.3389/fnins.2013.00055/abstract>

48.3.2 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]
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
high_pass_filter_cutoff: (a float)
    High-pass filter cutoff in secs
input_units: ('secs' or 'scans')

```

(continues on next page)

(continued from previous page)

```

        Units of event onsets and durations (secs or scans). Output units
        are always in secs
time_repetition: (a float)
    Time between the start of one volume to the start of the next image
    volume.
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
time_acquisition: (a float)
    Time in seconds to acquire a single image volume
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

[Optional]
model_hrf: (a boolean)
    Model sparse events with hrf
volumes_in_cluster: (a long integer >= 1, nipy default value: 1)
    Number of scan volumes in a cluster
scale_regressors: (a boolean, nipy default value: True)
    Scale regressors by the peak
use_temporal_deriv: (a boolean)
    Create a temporal derivative in addition to regular regressor
    requires: model_hrf
stimuli_as_impulses: (a boolean, nipy default value: True)
    Treat each stimulus to be impulse-like
save_plot: (a boolean)
    Save plot of sparse design calculation (requires matplotlib)
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
scan_onset: (a float, nipy default value: 0.0)
    Start of scanning relative to onset of run in secs
outlier_files: (a list of items which are an existing file name)
    Files containing scan outlier indices that should be tossed

```

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

```

48.4 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)
~~
```

48.5 gen_info()

[Link to code](#)

Generate subject_info structure from a list of event files

48.6 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
```

48.7 scale_timings()

[Link to code](#)

Scales timings given input and output units (scans/secs)

48.7.1 Parameters

timelist: list of times to scale input_units: 'secs' or 'scans' output_units: Ibid. time_repetition: float in seconds

48.8 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
```

(continues on next page)

(continued from previous page)

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]			

49.1 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. If you wish to use individual thresholds instead, import *Undefined* from *nipype.interfaces.base* and set *...inputs.use_norm = Undefined*

49.1.1 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]
realigned_files: (a list of items which are an existing file name)
    Names of realigned functional data files
rotation_threshold: (a float)
    Threshold (in radians) to use to detect rotation-related outliers
mutually_exclusive: norm_threshold
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
```

(continues on next page)

(continued from previous page)

```

zintensity_threshold: (a float)
    Intensity Z-threshold use to detection images that deviate from the
    mean
realignment_parameters: (a list of items which are an existing file
    name)
    Names of realignment parameters corresponding to the functional data
    files
parameter_source: ('SPM' or 'FSL' or 'AFNI' or 'NiPy' or 'FSFAST')
    Source of movement parameters
translation_threshold: (a float)
    Threshold (in mm) to use to detect translation-related outliers
mutually_exclusive: norm_threshold
norm_threshold: (a float)
    Threshold to use to detect motion-related outliers when composite
    motion is being used
mutually_exclusive: rotation_threshold, translation_threshold

[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
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 parameter (second element) estimates in order to determine
    outliers. (default is [True, False])
mask_file: (an existing file name)
    Mask file to be used if mask_type is 'file'.
intersect_mask: (a boolean, nipy default value: True)
    Intersect the masks when computed from spm_global.
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
mask_threshold: (a float)
    Mask threshold to be used if mask_type is 'thresh'.
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 voxel
    displacement timeseries
plot_files: (a list of items which are a file name)
    One image file for each functional run containing the detected
    outliers
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
statistic_files: (a list of items which are an existing file name)
    One file for each functional run containing information about the

```

(continues on next page)

(continued from previous page)

```

different types of artifacts and if design info is provided then
details of stimulus correlated motion and a listing or artifacts by
event type.
mask_files: (a list of items which are a file name)
    One image file for each functional run containing the mask used 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

```

49.2 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.

49.2.1 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]
spm_mat_file: (an existing file name)
    SPM mat file (use pre-estimate SPM.mat file)
concatenated_design: (a boolean)
    state if the design matrix contains concatenated sessions
realignment_parameters: (a list of items which are an existing file
    name)
    Names of realignment parameters corresponding to the functional data
    files
intensity_values: (a list of items which are an existing file name)
    Name of file containing intensity values

```

Outputs:

```

stimcorr_files: (a list of items which are an existing file name)
    List of files containing correlation values

```


50.1 ActivationCount

[Link to code](#)

Calculate a simple Activation Count Maps

Adapted from: https://github.com/poldracklab/CNP_task_analysis/blob/61c27f5992db9d8800884f8ffceb73e6957db8af/CNP_2n

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
           input file, generally a list of z-stat maps
threshold: (a float)
            binarization threshold. E.g. a threshold of 1.65 corresponds to a
            two-sided Z-test of  $p < .10$ 
```

Outputs:

```
acm_neg: (an existing file name)
          negative activation count map
out_file: (an existing file name)
          output activation count map
acm_pos: (an existing file name)
          positive activation count map
```


51.1 interfaces.afni.base

51.1.1 AFNI Python Command

[Link to code](#)

Inputs:

```
[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
num_threads: (an integer (int or long), nipy default value: 1)
            set number of threads
out_file: (a file name)
            output image file name
            argument: ``-prefix %s``
args: (a unicode string)
            Additional parameters to the command
            argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
            of class 'str' and with values which are a bytes or None or a
            value of class 'str', nipy default value: {})
            Environment variables
```

Outputs:

```
None
```

References:

None None

51.2 interfaces.afni.model

51.2.1 Deconvolve

[Link to code](#)

Wraps the executable command 3dDeconvolve.
 Performs OLS regression given a 4D neuroimage file and stimulus timings
 For complete details, see the [3dDeconvolve Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> deconvolve = afni.Deconvolve()
>>> deconvolve.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> deconvolve.inputs.out_file = 'output.nii'
>>> deconvolve.inputs.x1D = 'output.1D'
>>> stim_times = [(1, 'timeseries.txt', 'SPMG1(4)')]
>>> deconvolve.inputs.stim_times = stim_times
>>> deconvolve.inputs.stim_label = [(1, 'Houses')]
>>> deconvolve.inputs.gltsym = ['SYM: +Houses']
>>> deconvolve.inputs.glt_label = [(1, 'Houses')]
>>> deconvolve.cmdline
"3dDeconvolve -input functional.nii functional2.nii -bucket output.nii -x1D_
↪output.1D -num_stimts 1 -stim_times 1 timeseries.txt 'SPMG1(4)' -stim_label 1_
↪Houses -num_glt 1 -gltsym 'SYM: +Houses' -glt_label 1 Houses"
>>> res = deconvolve.run()
```

Inputs:

```
[Optional]
global_times: (a boolean)
    use global timing for stimulus timing files
    argument: ``-global_times``
    mutually_exclusive: local_times
cbucket: (a unicode string)
    Name for dataset in which to save the regression coefficients (no
    statistics). This dataset will be used in a -xrestore run [not yet
    implemented] instead of the bucket dataset, if possible.
    argument: ``-cbucket %s``
ortvec: (a tuple of the form: (an existing file name, a unicode
    string))
    this option lets you input a rectangular array of 1 or more baseline
    vectors from a file. This method is a fast way to include a lot of
    baseline regressors in one step.
    argument: ``-ortvec %s %s``
nofdr: (a boolean)
    Don't compute the statistic-vs-FDR curves for the bucket dataset.
    argument: ``-noFDR``
fout: (a boolean)
    output F-statistic for each stimulus
    argument: ``-fout``
stim_label: (a list of items which are a tuple of the form: (an
    integer (int or long), a unicode string))
    label for kth input stimulus (e.g., Label1)
    argument: ``-stim_label %d %s...``, position: -4
    requires: stim_times
num_stimts: (an integer (int or long))
    number of stimulus timing files
    argument: ``-num_stimts %d``, position: -6
rout: (a boolean)
    output the R^2 statistic for each stimulus
    argument: ``-rout``
```

(continues on next page)

(continued from previous page)

```

automask: (a boolean)
    build a mask automatically from input data (will be slow for long
    time series datasets)
    argument: ``-automask``
input1D: (an existing file name)
    filename of single (fMRI) .1D time series where time runs down the
    column.
    argument: ``-input1D %s``
in_files: (a list of items which are an existing file name)
    filenames of 3D+time input datasets. More than one filename can be
    given and the datasets will be auto-catenated in time. You can input
    a 1D time series file here, but the time axis should run along the
    ROW direction, not the COLUMN direction as in the 'input1D' option.
    argument: ``-input %s``, position: 1
trans: (a boolean)
    check the dataset time series for initial saturation transients,
    which should normally have been excised before data analysis.
    argument: ``-trans``
    mutually_exclusive: sat
dname: (a tuple of the form: (a unicode string, a unicode string))
    set environmental variable to provided value
    argument: ``-D%s=%s``
x1D_stop: (a boolean)
    stop running after writing .xmat.1D file
    argument: ``-x1D_stop``
legendre: (a boolean)
    use Legendre polynomials for null hypothesis (baseline model)
    argument: ``-legendre``
nocond: (a boolean)
    DON'T calculate matrix condition number
    argument: ``-nocond``
nolegendre: (a boolean)
    use power polynomials for null hypotheses. Don't do this unless you
    are crazy!
    argument: ``-nolegendre``
dmbase: (a boolean)
    de-mean baseline time series (default if 'polort' >= 0)
    argument: ``-dmbase``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
force_TR: (a float)
    use this value instead of the TR in the 'input' dataset. (It's
    better to fix the input using Refit.)
    argument: ``-force_TR %f``, position: 0
x1D: (a file name)
    specify name for saved X matrix
    argument: ``-x1D %s``
polort: (an integer (int or long))
    degree of polynomial corresponding to the null hypothesis [default:
    1]
    argument: ``-polort %d``
gltsym: (a list of items which are a unicode string)
    general linear tests (i.e., contrasts) using symbolic conventions
    (e.g., '+Label1 -Label2')
    argument: ``-gltsym 'SYM: %s'...``, position: -2
TR_1D: (a float)
    TR to use with 'input1D'. This option has no effect if you do not

```

(continues on next page)

(continued from previous page)

```

    also use 'input1D'.
    argument: ``-TR_1D %f``
STATmask: (an existing file name)
    build a mask from provided file, and use this mask for the purpose
    of reporting truncation-to float issues AND for computing the FDR
    curves. The actual results ARE not masked with this option (only
    with 'mask' or 'automask' options).
    argument: ``-STATmask %s``
singvals: (a boolean)
    print out the matrix singular values
    argument: ``-singvals``
nodmbase: (a boolean)
    don't de-mean baseline time series
    argument: ``-nodmbase``
stim_times_subtract: (a float)
    this option means to subtract specified seconds from each time
    encountered in any 'stim_times' option. The purpose of this option
    is to make it simple to adjust timing files for the removal of
    images from the start of each imaging run.
    argument: ``-stim_times_subtract %f``
stim_times: (a list of items which are a tuple of the form: (an
    integer (int or long), an existing file name, a unicode string))
    generate a response model from a set of stimulus times given in
    file.
    argument: ``-stim_times %d %s '%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
glt_label: (a list of items which are a tuple of the form: (an
    integer (int or long), a unicode string))
    general linear test (i.e., contrast) labels
    argument: ``-glt_label %d %s...``, position: -1
    requires: gltsym
noblock: (a boolean)
    normally, if you input multiple datasets with 'input', then the
    separate datasets are taken to be separate image runs that get
    separate baseline models. Use this options if you want to have the
    program consider these to be all one big run.* If any of the input
    dataset has only 1 sub-brick, then this option is automatically
    invoked!* If the auto-catenation feature isn't used, then this
    option has no effect, no how, no way.
    argument: ``-noblock``
local_times: (a boolean)
    use local timing for stimulus timing files
    argument: ``-local_times``
    mutually_exclusive: global_times
out_file: (a file name)
    output statistics file
    argument: ``-bucket %s``
mask: (an existing file name)
    filename of 3D mask dataset; only data time series from within the
    mask will be analyzed; results for voxels outside the mask will be
    set to zero.
    argument: ``-mask %s``
svd: (a boolean)
    use SVD instead of Gaussian elimination (default)

```

(continues on next page)

(continued from previous page)

```

        argument: ``-svd``
num_glt: (an integer (int or long))
        number of general linear tests (i.e., contrasts)
        argument: ``-num_glt %d``, position: -3
sat: (a boolean)
        check the dataset time series for initial saturation transients,
        which should normally have been excised before data analysis.
        argument: ``-sat``
        mutually_exclusive: trans
tout: (a boolean)
        output the T-statistic for each stimulus
        argument: ``-tout``
num_threads: (an integer (int or long))
        run the program with provided number of sub-processes
        argument: ``-jobs %d``
vout: (a boolean)
        output the sample variance (MSE) for each stimulus
        argument: ``-vout``
nosvd: (a boolean)
        use Gaussian elimination instead of SVD
        argument: ``-nosvd``
allzero_OK: (a boolean)
        don't consider all zero matrix columns to be the type of error that
        'gotforit' is needed to ignore.
        argument: ``-allzero_OK``
censor: (an existing file name)
        filename of censor .1D time series. This is a file of 1s and 0s,
        indicating which time points are to be included (1) and which are to
        be excluded (0).
        argument: ``-censor %s``
rmsmin: (a float)
        minimum rms error to reject reduced model (default = 0; don't use
        this option normally!)
        argument: ``-rmsmin %f``
goforit: (an integer (int or long))
        use this to proceed even if the matrix has bad problems (e.g.,
        duplicate columns, large condition number, etc.).
        argument: ``-GOFORIT %i``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``

```

Outputs:

```

reml_script: (an existing file name)
        automatical generated script to run 3dREMLfit
out_file: (an existing file name)
        output statistics file
cbucket: (a file name)
        output regression coefficients file (if generated)
x1D: (an existing file name)
        save out X matrix

```

References:

None None

51.2.2 Remlfit

[Link to code](#)

Wraps the executable command 3dREMLfit.

Performs Generalized least squares time series fit with Restricted Maximum Likelihood (REML) estimation of the temporal auto-correlation structure.

For complete details, see the [3dREMLfit Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> remlfit = afni.Remlfit()
>>> remlfit.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> remlfit.inputs.out_file = 'output.nii'
>>> remlfit.inputs.matrix = 'output.1D'
>>> remlfit.inputs.gltsym = [('SYM: +Lab1 -Lab2', 'TestSYM'), ('timeseries.txt',
    ↪ 'TestFile')]
>>> remlfit.cmdline
'3dREMLfit -gltsym "SYM: +Lab1 -Lab2" TestSYM -gltsym "timeseries.txt" TestFile -
    ↪input "functional.nii functional2.nii" -matrix output.1D -Rbuck output.nii'
>>> res = remlfit.run()
```

Inputs:

```
[Mandatory]
matrix: (a file name)
    the design matrix file, which should have been output from
    Deconvolve via the 'x1D' option
    argument: ``-matrix %s``
in_files: (a list of items which are an existing file name)
    Read time series dataset
    argument: ``-input "%s"``

[Optional]
oerrts: (a file name)
    dataset for OLSQ residuals (data - fitted model)
    argument: ``-Oerrts %s``
wherr_file: (a file name)
    dataset for REML residual, whitened using the estimated ARMA(1,1)
    correlation matrix of the noise
    argument: ``-Rwherr %s``
ofitts: (a file name)
    dataset for OLSQ fitted model
    argument: ``-Ofitts %s``
nofdr: (a boolean)
    do NOT add FDR curve data to bucket datasets; FDR curves can take a
    long time if 'tout' is used
    argument: ``-noFDR``
errts_file: (a file name)
    output dataset for REML residuals = data - fitted model
    argument: ``-Rerrts %s``
glt_file: (a file name)
    output dataset for beta + statistics from the REML estimation, but
    ONLY for the GLTs added on the REMLfit command line itself via
    'gltsym'; GLTs from Deconvolve's command line will NOT be included.
    argument: ``-Rglt %s``
fout: (a boolean)
    output F-statistic for each stimulus
```

(continues on next page)

(continued from previous page)

```

        argument: ``-fout``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
oglt: (a file name)
        dataset for beta + statistics from 'gltsym' options
        argument: ``-Oglts %s``
automask: (a boolean, nipy default value: False)
        build a mask automatically from input data (will be slow for long
        time series datasets)
        argument: ``-automask``
addbase: (a list of items which are an existing file name)
        file(s) to add baseline model columns to the matrix with this
        option. Each column in the specified file(s) will be appended to the
        matrix. File(s) must have at least as many rows as the matrix does.
        argument: ``-addbase %s``
var_file: (a file name)
        output dataset for REML variance parameters
        argument: ``-Rvar %s``
rout: (a boolean)
        output the R^2 statistic for each stimulus
        argument: ``-rout``
matim: (a file name)
        read a standard file as the matrix. You can use only Col as a name
        in GLTs with these nonstandard matrix input methods, since the other
        names come from the 'matrix' file. These mutually exclusive options
        are ignored if 'matrix' is used.
        argument: ``-matim %s``
        mutually_exclusive: matrix
slibase_sm: (a list of items which are an existing file name)
        similar to 'slibase', BUT each file must be in slice major order
        (i.e. all slice0 columns come first, then all slice1 columns, etc).
        argument: ``-slibase_sm %s``
fitts_file: (a file name)
        output dataset for REML fitted model
        argument: ``-Rfitts %s``
STATmask: (an existing file name)
        filename of 3D mask dataset to be used for the purpose of reporting
        truncation-to float issues AND for computing the FDR curves. The
        actual results ARE not masked with this option (only with 'mask' or
        'automask' options).
        argument: ``-STATmask %s``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
nobout: (a boolean)
        do NOT add baseline (null hypothesis) regressor betas to the
        'rbeta_file' and/or 'obeta_file' output datasets.
        argument: ``-nobout``
out_file: (a file name)
        output dataset for beta + statistics from the REML estimation; also
        contains the results of any GLT analysis requested in the Deconvolve
        setup, similar to the 'bucket' output from Deconvolve. This dataset
        does NOT get the betas (or statistics) of those regressors marked as
        'baseline' in the matrix file.
        argument: ``-Rbuck %s``
obuck: (a file name)

```

(continues on next page)

(continued from previous page)

```

    dataset for beta + statistics from the OLSQ estimation
    argument: ``-Obuck %s``
tout: (a boolean)
    output the T-statistic for each stimulus; if you use 'out_file' and
    do not give any of 'fout', 'tout', or 'rout', then the program
    assumes 'fout' is activated.
    argument: ``-tout``
dsort: (an existing file name)
    4D dataset to be used as voxelwise baseline regressor
    argument: ``-dsort %s``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
nodmbase: (a boolean)
    by default, baseline columns added to the matrix via 'addbase' or
    'slibase' or 'dsort' will each have their mean removed (as is done
    in Deconvolve); this option turns this centering off
    argument: ``-nodmbase``
    requires: addbase, dsort
verb: (a boolean)
    turns on more progress messages, including memory usage progress
    reports at various stages
    argument: ``-verb``
rbeta_file: (a file name)
    output dataset for beta weights from the REML estimation, similar to
    the 'cbucket' output from Deconvolve. This dataset will contain all
    the beta weights, for baseline and stimulus regressors alike, unless
    the '-nobout' option is given -- in that case, this dataset will
    only get the betas for the stimulus regressors.
    argument: ``-Rbeta %s``
gltsym: (a list of items which are a tuple of the form: (an existing
    file name, a unicode string) or a tuple of the form: (a unicode
    string, a unicode string))
    read a symbolic GLT from input file and associate it with a label.
    As in Deconvolve, you can also use the 'SYM:' method to provide the
    definition of the GLT directly as a string (e.g., with 'SYM: +Label1
    -Label2'). Unlike Deconvolve, you MUST specify 'SYM: ' if providing
    the GLT directly as a string instead of from a file
    argument: ``-gltsym "%s" %s...``
ovar: (a file name)
    dataset for OLSQ st.dev. parameter (kind of boring)
    argument: ``-Ovar %s``
mask: (an existing file name)
    filename of 3D mask dataset; only data time series from within the
    mask will be analyzed; results for voxels outside the mask will be
    set to zero.
    argument: ``-mask %s``
obeta: (a file name)
    dataset for beta weights from the OLSQ estimation
    argument: ``-Obeta %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
dsort_nods: (a boolean)
    if 'dsort' option is used, this command will output additional
    results files excluding the 'dsort' file
    argument: ``-dsort_nods``
    requires: dsort

```

(continues on next page)

(continued from previous page)

```

polort: (an integer (int or long))
    if no 'matrix' option is given, AND no 'matim' option, create a
    matrix with Legendre polynomial regressors up to the specified order.
    The default value is 0, which produces a matrix with a single column
    of all ones
    argument: ``-polort %d``
    mutually_exclusive: matrix
quiet: (a boolean)
    turn off most progress messages
    argument: ``-quiet``
slibase: (a list of items which are an existing file name)
    similar to 'addbase' in concept, BUT each specified file must have
    an integer multiple of the number of slices in the input dataset(s);
    then, separate regression matrices are generated for each slice,
    with the first column of the file appended to the matrix for the
    first slice of the dataset, the second column of the file appended
    to the matrix for the first slice of the dataset, and so on.
    Intended to help model physiological noise in FMRI, or other effects
    you want to regress out that might change significantly in the
    inter-slice time intervals. This will slow the program down, and
    make it use a lot more memory (to hold all the matrix stuff).
    argument: ``-slibase %s``
usetemp: (a boolean)
    write intermediate stuff to disk, to economize on RAM. Using this
    option might be necessary to run with 'slibase' and with 'Grid'
    values above the default, since the program has to store a large
    number of matrices for such a problem: two for every slice and for
    every (a,b) pair in the ARMA parameter grid. Temporary files are
    written to the directory given in environment variable TMPDIR, or in
    /tmp, or in ./ (preference is in that order)
    argument: ``-usetemp``

```

Outputs:

```

oertrs: (a file name)
    dataset for OLSQ residuals = data - fitted model (if generated
wherr_file: (a file name)
    dataset for REML residual, whitened using the estimated ARMA(1,1)
    correlation matrix of the noise (if generated)
fitts_file: (a file name)
    output dataset for REML fitted model (if generated)
ofitts: (a file name)
    dataset for OLSQ fitted model (if generated)
ertrs_file: (a file name)
    output dataset for REML residuals = data - fitted model (if
    generated)
var_file: (a file name)
    dataset for REML variance parameters (if generated)
glt_file: (a file name)
    output dataset for beta + statistics from the REML estimation, but
    ONLY for the GLTs added on the REMLfit command line itself via
    'gltsym' (if generated)
ovar: (a file name)
    dataset for OLSQ st.dev. parameter (if generated)
obuck: (a file name)
    dataset for beta + statistics from the OLSQ estimation (if
    generated)

```

(continues on next page)

(continued from previous page)

```

obeta: (a file name)
        dataset for beta weights from the OLSQ estimation (if generated)
oglt: (a file name)
        dataset for beta + statistics from 'gltsym' options (if generated)
rbeta_file: (a file name)
        output dataset for beta weights from the REML estimation (if
        generated)
out_file: (a file name)
        dataset for beta + statistics from the REML estimation (if generated

```

References:

None None

51.2.3 Synthesize

[Link to code](#)

Wraps the executable command 3dSynthesize.

Reads a ‘-cbucket’ dataset and a ‘.xmat.1D’ matrix from 3dDeconvolve, and synthesizes a fit dataset using user-selected sub-bricks and matrix columns.

For complete details, see the [3dSynthesize Documentation](#).

Examples

```

>>> from nipy.interfaces import afni
>>> synthesize = afni.Synthesize()
>>> synthesize.inputs.cbucket = 'functional.nii'
>>> synthesize.inputs.matrix = 'output.1D'
>>> synthesize.inputs.select = ['baseline']
>>> synthesize.cmdline
'3dSynthesize -cbucket functional.nii -matrix output.1D -select baseline'
>>> syn = synthesize.run()

```

Inputs:

```

[Mandatory]
select: (a list of items which are a unicode string)
        A list of selected columns from the matrix (and the corresponding
        coefficient sub-bricks from the cbucket). Valid types include
        'baseline', 'polort', 'allfunc', 'allstim', 'all', Can also provide
        'something' where something matches a stim_label from 3dDeconvolve,
        and 'digits' where digits are the numbers of the select matrix
        columns by numbers (starting at 0), or number ranges of the form
        '3..7' and '3-7'.
        argument: ``-select %s``
matrix: (a file name)
        Read the matrix output from 3dDeconvolve via the '-x1D' option.
        argument: ``-matrix %s``
cbucket: (a file name)
        Read the dataset output from 3dDeconvolve via the '-cbucket' option.
        argument: ``-cbucket %s``

[Optional]
cenfill: ('zero' or 'nbhr' or 'none')
        Determines how censored time points from the 3dDeconvolve run will

```

(continues on next page)

(continued from previous page)

```

        be filled. Valid types are 'zero', 'nbhr' and 'none'.
        argument: ``-cenfill %s``
dry_run: (a boolean)
        Don't compute the output, just check the inputs.
        argument: ``-dry``
out_file: (a file name)
        output dataset prefix name (default 'syn')
        argument: ``-prefix %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
        set number of threads
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
TR: (a float)
        TR to set in the output. The default value of TR is read from the
        header of the matrix file.
        argument: ``-TR %f``

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:

None None

51.3 interfaces.afni.preprocess

51.3.1 AlignEpiAnatPy

[Link to code](#)

Wraps the executable command `align_epi_anat.py`.

Align EPI to anatomical datasets or vice versa This Python script computes the alignment between two datasets, typically an EPI and an anatomical structural dataset, and applies the resulting transformation to one or the other to bring them into alignment.

This script computes the transforms needed to align EPI and anatomical datasets using a cost function designed for this purpose. The script combines multiple transformations, thereby minimizing the amount of interpolation applied to the data.

Basic Usage: `align_epi_anat.py -anat anat+orig -epi epi+orig -epi_base 5`

The user must provide EPI and anatomical datasets and specify the EPI sub-brick to use as a base in the alignment.

Internally, the script always aligns the anatomical to the EPI dataset, and the resulting transformation is saved to a 1D file. As a user option, the inverse of this transformation may be applied to the EPI dataset in order to align it to the anatomical data instead.

This program generates several kinds of output in the form of datasets and transformation matrices which can be applied to other datasets if needed. Time-series volume registration, oblique data transformations and Talairach (standard template) transformations will be combined as needed and requested (with options to turn on and off each of the steps) in order to create the aligned datasets.

For complete details, see the `align_epi_anat.py` Documentation.

Examples

```
>>> from nipyype.interfaces import afni
>>> al_ea = afni.AlignEpiAnatPy()
>>> al_ea.inputs.anat = "structural.nii"
>>> al_ea.inputs.in_file = "functional.nii"
>>> al_ea.inputs.epi_base = 0
>>> al_ea.inputs.epi_strip = '3dAutomask'
>>> al_ea.inputs.volreg = 'off'
>>> al_ea.inputs.tshift = 'off'
>>> al_ea.inputs.save_skullstrip = True
>>> al_ea.cmdline
'python2 ...align_epi_anat.py -anat structural.nii -epi_base 0 -epi_strip_
↪3dAutomask -epi functional.nii -save_skullstrip -suffix _al -tshift off -volreg_
↪off'
>>> res = allineate.run()
```

Inputs:

```
[Mandatory]
epi_base: (a long integer >= 0 or 'mean' or 'median' or 'max')
    the epi base used in alignment should be one of
    (0/mean/median/max/subbrick#)
    argument: ``-epi_base %s``
anat: (an existing file name)
    name of structural dataset
    argument: ``-anat %s``
in_file: (an existing file name)
    EPI dataset to align
    argument: ``-epi %s``

[Optional]
suffix: (a unicode string, nipyype default value: _al)
    append suffix to the original anat/epi dataset to use in the
    resulting dataset names (default is "_al")
    argument: ``-suffix %s``
py27_path: (an existing file name or 'python2', nipyype default value:
    python2)
volreg: ('on' or 'off', nipyype default value: on)
    do volume registration on EPI dataset before alignment should be 'on'
    or 'off', defaults to 'on'
    argument: ``-volreg %s``
tshift: ('on' or 'off', nipyype default value: on)
    do time shifting of EPI dataset before alignment should be 'on' or
    'off', defaults to 'on'
    argument: ``-tshift %s``
epi_strip: ('3dSkullStrip' or '3dAutomask' or 'None')
    method to mask brain in EPI data should be one
    of [3dSkullStrip]/3dAutomask/None)
    argument: ``-epi_strip %s``
anat2epi: (a boolean)
    align anatomical to EPI dataset (default)
    argument: ``-anat2epi``
environ: (a dictionary with keys which are a 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: {})
```

(continues on next page)

(continued from previous page)

```

    Environment variables
save_skullstrip: (a boolean)
    save skull-stripped (not aligned)
    argument: ``-save_skullstrip``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
epi2anat: (a boolean)
    align EPI to anatomical dataset
    argument: ``-epi2anat``

```

Outputs:

```

anat_al_mat: (a file name)
    matrix to align anatomy to the EPI
anat_al_orig: (a file name)
    A version of the anatomy that is aligned to the EPI
skullstrip: (a file name)
    skull-stripped (not aligned) volume
epi_vr_motion: (a file name)
    motion parameters from EPI time-seriesregistration (tsh included in
    name if slicetiming correction is also included).
epi_vr_al_mat: (a file name)
    matrix to volume register EPI
epi_reg_al_mat: (a file name)
    matrix to volume register and align epi to anatomy
epi_al_tlrc_mat: (a file name)
    matrix to volume register and align epito anatomy and put into
    standard space
epi_al_orig: (a file name)
    A version of the EPI dataset aligned to the anatomy
epi_al_mat: (a file name)
    matrix to align EPI to anatomy
epi_tlrc_al: (a file name)
    A version of the EPI dataset aligned to a standard template

```

References:

None None

51.3.2 Allineate[Link to code](#)

Wraps the executable 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'

```

(continues on next page)

(continued from previous page)

```
>>> allineate.inputs.in_matrix = 'cmatrix.mat'
>>> allineate.cmdline
'3dAllineate -source functional.nii -prefix functional_allineate.nii -lDmatrix_
↳apply cmatrix.mat'
>>> res = allineate.run()
```

```
>>> allineate = afni.Allineate()
>>> allineate.inputs.in_file = 'functional.nii'
>>> allineate.inputs.reference = 'structural.nii'
>>> allineate.inputs.allcostx = 'out.allcostX.txt'
>>> allineate.cmdline
'3dAllineate -source functional.nii -base structural.nii -allcostx |& tee out.
↳allcostX.txt'
>>> res = allineate.run()
```

```
>>> allineate = afni.Allineate()
>>> allineate.inputs.in_file = 'functional.nii'
>>> allineate.inputs.reference = 'structural.nii'
>>> allineate.inputs.nwarp_fixmot = ['X', 'Y']
>>> allineate.cmdline
'3dAllineate -source functional.nii -nwarp_fixmotX -nwarp_fixmotY -prefix_
↳functional_allineate -base structural.nii'
>>> res = allineate.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input file to 3dAllineate
        argument: ``-source %s``

[Optional]
two_blur: (a float)
        Set the blurring radius for the first pass in mm.
        argument: ``-twoblur %f``
nmatch: (an integer (int or long))
        Use at most n scattered points to match the datasets.
        argument: ``-nmatch %d``
nomask: (a boolean)
        Don't compute the autoweight/mask; if -weight is not also used, then
        every voxel will be counted equally.
        argument: ``-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.
        argument: ``-nwarp %s``
replacebase: (a boolean)
        If the source has more than one volume, then after the first volume
        is aligned to the base.
        argument: ``-replacebase``
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.
        argument: ``-nwarp_fixdep%s...``
maxscl: (a float)
        Maximum allowed scaling factor.
```

(continues on next page)

(continued from previous page)

```

    argument: ``-maxscl %f``
in_param_file: (an existing file name)
    Read warp parameters from file and apply them to the source dataset,
    and produce a new dataset
    argument: ``-1Dparam_apply %s``
    mutually_exclusive: out_param_file
warp_type: ('shift_only' or 'shift_rotate' or 'shift_rotate_scale' or
    'affine_general')
    Set the warp type.
    argument: ``-warp %s``
newgrid: (a float)
    Write the output dataset using isotropic grid spacing in mm.
    argument: ``-newgrid %f``
final_interpolation: ('nearestneighbour' or 'linear' or 'cubic' or
    'quintic' or 'wsinc5')
    Defines interpolation method used to create the output dataset
    argument: ``-final %s``
automask: (an integer (int or long))
    Compute a mask function, set a value for dilation or 0.
    argument: ``-automask+%d``
out_param_file: (a file name)
    Save the warp parameters in ASCII (.1D) format.
    argument: ``-1Dparam_save %s``
    mutually_exclusive: in_param_file, allcostx
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
    argument: ``-weight %s``
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'.
    argument: ``-replacemeth %s``
warpfreeze: (a boolean)
    Freeze the non-rigid body parameters after first volume.
    argument: ``-warpfreeze``
nwarp_fixmot: (a list of items which are 'X' or 'Y' or 'Z' or 'I' or
    'J' or 'K')
    To fix motion along directions.
    argument: ``-nwarp_fixmot%s...``
zclip: (a boolean)
    Replace negative values in the input datasets (source & base) with
    zero.
    argument: ``-zclip``
maxrot: (a float)
    Maximum allowed rotation in degrees.
    argument: ``-maxrot %f``
master: (an existing file name)
    Write the output dataset on the same grid as this file.
    argument: ``-master %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

```

(continues on next page)

(continued from previous page)

```

        volume is very noisy. [Default == 0 mm = no blurring at the final
        alignment pass]
        argument: ``-fineblur %f``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
no_pad: (a boolean)
        Do not use zero-padding on the base image.
        argument: ``-nopad``
convergence: (a float)
        Convergence test in millimeters (default 0.05mm).
        argument: ``-conv %f``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
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.
        argument: ``-EPI``
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.
        argument: ``-check %s``
quiet: (a boolean)
        Don't print out verbose progress reports.
        argument: ``-quiet``
source_mask: (an existing file name)
        mask the input dataset
        argument: ``-source_mask %s``
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.
        argument: ``-twofirst``
maxshf: (a float)
        Maximum allowed shift in mm.
        argument: ``-maxshf %f``
interpolation: ('nearestneighbour' or 'linear' or 'cubic' or
        'quintic')
        Defines interpolation method to use during matching
        argument: ``-interp %s``
allcostx: (a file name)
        Compute and print ALL available cost functionals for the un-warped
        inputs AND THEN QUIT. If you use this option none of the other
        expected outputs will be produced
        argument: ``-allcostx |& tee %s``, position: -1
        mutually_exclusive: out_file, out_matrix, out_param_file,
        out_weight_file
two_pass: (a boolean)
        Use a two pass alignment strategy for all volumes, searching for a

```

(continues on next page)

(continued from previous page)

```

        large rotation+shift and then refining the alignment.
        argument: ``-twopass``
environ: (a dictionary with keys which are a 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
center_of_mass: (a unicode string)
        Use the center-of-mass calculation to bracket the shifts.
        argument: ``-cmass%s``
source_automask: (an integer (int or long))
        Automatically mask the source dataset with dilation or 0.
        argument: ``-source_automask+%d``
usetemp: (a boolean)
        temporary file use
        argument: ``-usetemp``
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
        argument: ``-cost %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.
        argument: ``-onepass``
autobox: (a boolean)
        Expand the -automask function to enclose a rectangular box that
        holds the irregular mask.
        argument: ``-autobox``
out_matrix: (a file name)
        Save the transformation matrix for each volume.
        argument: ``-lDmatrix_save %s``
        mutually_exclusive: in_matrix, allcostx
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.
        argument: ``-base %s``
maxshr: (a float)
        Maximum allowed shearing factor.
        argument: ``-maxshr %f``
num_threads: (an integer (int or long), nipyte default value: 1)
        set number of threads
in_matrix: (a file name)
        matrix to align input file
        argument: ``-lDmatrix_apply %s``, position: -3
        mutually_exclusive: out_matrix
out_weight_file: (a file name)
        Write the weight volume to disk as a dataset
        argument: ``-wtprefix %s``
        mutually_exclusive: allcostx
weight: (an existing file name or a float)
        Set the weighting for each voxel in the base dataset; larger weights
        mean that voxel count more in the cost function. If an image file is
        given, the volume must be defined on the same grid as the base
        dataset
        argument: ``-weight %s``

```

(continues on next page)

(continued from previous page)

```

autoweight: (a unicode string)
    Compute a weight function using the 3dAutomask algorithm plus some
    blurring of the base image.
    argument: ``-autoweight%s``
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]
    argument: ``-twobest %d``
verbose: (a boolean)
    Print out verbose progress reports.
    argument: ``-verb``
overwrite: (a boolean)
    overwrite output file if it already exists
    argument: ``-overwrite``
out_file: (a file name)
    output file from 3dAllineate
    argument: ``-prefix %s``
mutually_exclusive: allcostx

```

Outputs:

```

out_param_file: (an existing file name)
    warp parameters
out_matrix: (an existing file name)
    matrix to align input file
out_weight_file: (an existing file name)
    weight volume
out_file: (an existing file name)
    output image file name
allcostx: (a file name)
    Compute and print ALL available cost functionals for the un-warped
    inputs

```

References:

None None

51.3.3 AutoTLRC[Link to code](#)

Wraps the executable command @auto_tlrc.

A minmal wrapper for the AutoTLRC script The only option currently supported is no_ss. For complete details, see the [3dQwarp Documenta**tion**](#).

Examples

```

>>> from nipy.interfaces import afni
>>> autoTLRC = afni.AutoTLRC()
>>> autoTLRC.inputs.in_file = 'structural.nii'
>>> autoTLRC.inputs.no_ss = True
>>> autoTLRC.inputs.base = "TT_N27+tlrc"
>>> autoTLRC.cmdline
'@auto_tlrc -base TT_N27+tlrc -input structural.nii -no_ss'
>>> res = autoTLRC.run()

```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    Original anatomical volume (+orig).The skull is removed by this
    script unless instructed otherwise (-no_ss).
    argument: ``-input %s``
base: (a unicode string)
    Reference anatomical volume Usually this volume is in some standard
    space like TLRC or MNI space and with afni dataset view of (+tlrc).
    Preferably, this reference volume should have had the skull removed
    but that is not mandatory. AFNI's distribution contains several
    templates. For a longer list, use "whereami
    -show_templates"TT_N27+tlrc --> Single subject, skull stripped
    volume. This volume is also known as N27_SurfVol_NoSkull+tlrc
    elsewhere in AFNI and SUMA land. (www.loni.ucla.edu,
    www.bic.mni.mcgill.ca) This template has a full set of FreeSurfer
    (surfer.nmr.mgh.harvard.edu) surface models that can be used in
    SUMA. For details, see Talairach-related link:
    https://afni.nimh.nih.gov/afni/sumaTT_icbm452+tlrc --> Average
    volume of 452 normal brains. Skull Stripped.
    (www.loni.ucla.edu)TT_avg152T1+tlrc --> Average volume of 152 normal
    brains. Skull Stripped. (www.bic.mni.mcgill.ca)TT_EPI+tlrc --> EPI
    template from spm2, masked as TT_avg152T1 TT_avg152 and TT_EPI
    volume sources are from SPM's distribution.
    (www.fil.ion.ucl.ac.uk/spm/)If you do not specify a path for the
    template, the script will attempt to locate the template AFNI's
    binaries directory.NOTE: These datasets have been slightly modified
    from their original size to match the standard TLRC dimensions (Jean
    Talairach and Pierre Tournoux Co-Planar Stereotaxic Atlas of the
    Human Brain Thieme Medical Publishers, New York, 1988). That was
    done for internal consistency in AFNI. You may use the original form
    of these volumes if you choose but your TLRC coordinates will not be
    consistent with AFNI's TLRC database (San Antonio Talairach Daemon
    database), for example.
    argument: ``-base %s``

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
no_ss: (a boolean)
    Do not strip skull of input data set (because skull has already been
    removed or because template still has the skull)NOTE: The -no_ss
    option is not all that optional. Here is a table of when you should
    and should not use -no_ss Template Template WITH skull WITHOUT skull
    Dset. WITH skull -no_ss xxx WITHOUT skull No Cigar -no_ss Template
    means: Your template of choice Dset. means: Your anatomical dataset
    -no_ss means: Skull stripping should not be attempted on Dset xxx
    means: Don't put anything, the script will strip Dset No Cigar
    means: Don't try that combination, it makes no sense.
    argument: ``-no_ss``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.3.4 AutoTcorrelate

[Link to code](#)

Wraps the executable 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
         argument: ``%s``, position: -1

[Optional]
mask: (an existing file name)
      mask of voxels
      argument: ``-mask %s``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
eta2: (a boolean)
      eta^2 similarity
      argument: ``-eta2``
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
mask_only_targets: (a boolean)
                  use mask only on targets voxels
                  argument: ``-mask_only_targets``
                  mutually_exclusive: mask_source
```

(continues on next page)

(continued from previous page)

```

mask_source: (an existing file name)
    mask for source voxels
    argument: ``-mask_source %s``
    mutually_exclusive: mask_only_targets
polort: (an integer (int or long))
    Remove polynomial trend of order m or -1 for no detrending
    argument: ``-polort %d``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.3.5 Automask

[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
erode: (an integer (int or long))
    erode the mask inwards
    argument: ``-erode %s``
brain_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

        output file from 3dAutomask
        argument: ``-apply_prefix %s``
out_file: (a file name)
        output image file name
        argument: ``-prefix %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%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].
        argument: ``-clfrac %s``
num_threads: (an integer (int or long), nipy default value: 1)
        set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
dilate: (an integer (int or long))
        dilate the mask outwards
        argument: ``-dilate %s``

```

Outputs:

```

brain_file: (an existing file name)
        brain file (skull stripped)
out_file: (an existing file name)
        mask file

```

References:

None None

51.3.6 Bandpass[Link to code](#)

Wraps the executable 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]
lowpass: (a float)
    lowpass
    argument: ``%f``, position: -2
in_file: (an existing file name)
    input file to 3dBandpass
    argument: ``%s``, position: -1
highpass: (a float)
    highpass
    argument: ``%f``, position: -3

[Optional]
mask: (an existing file name)
    mask file
    argument: ``-mask %s``, position: 2
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.
    argument: ``-nodetrend``
out_file: (a file name)
    output file from 3dBandpass
    argument: ``-prefix %s``, position: 1
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.
    argument: ``-despike``
nfft: (an integer (int or long))
    Set the FFT length [must be a legal value].
    argument: ``-nfft %d``
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.
    argument: ``-ort %s``
blur: (a float)
    Blur (inside the mask only) with a filter width (FWHM) of 'fff'
    millimeters.
    argument: ``-blur %f``
normalize: (a boolean)
    Make all output time series have L2 norm = 1 (i.e., sum of squares =
    1).
    argument: ``-norm``
tr: (a float)
    Set time step (TR) in sec [default=from dataset header].
    argument: ``-dt %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
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.
    argument: ``-dsort %s``
notrans: (a boolean)
    Don't check for initial positive transients in the data. The test is

```

(continues on next page)

(continued from previous page)

```

        a little slow, so skipping it is OK, if you KNOW the data time
        series are transient-free.
        argument: ``-notrans``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
        set number of threads
automask: (a boolean)
        Create a mask from the input dataset.
        argument: ``-automask``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
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.
        argument: ``-localPV %f``

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:

None None

51.3.7 BlurInMask[Link to code](#)

Wraps the executable 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]
in_file: (an existing file name)
        input file to 3dSkullStrip
        argument: ``-input %s``, position: 1
fwhm: (a float)
        fwhm kernel size

```

(continues on next page)

(continued from previous page)

```

        argument: ``-FWHM %f``

[Optional]
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.
    argument: ``-mask %s``
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.
    argument: ``-preserve``
out_file: (a file name)
    output to the file
    argument: ``-prefix %s``, position: -1
float_out: (a boolean)
    Save dataset as floats, no matter what the input data type is.
    argument: ``-float``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
options: (a unicode string)
    options
    argument: ``%s``, position: 2
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
multimask: (a file name)
    Multi-mask dataset -- each distinct nonzero value in dataset will be
    treated as a separate mask for blurring purposes.
    argument: ``-Mmask %s``
automask: (a boolean)
    Create an automask from the input dataset.
    argument: ``-automask``

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.3.8 BlurToFWHM[Link to code](#)

Wraps the executable 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
        argument: ``-input %s``

[Optional]
mask: (an existing file name)
      Mask dataset, if desired. Voxels NOT in mask will be set to zero in
      output.
      argument: ``-mask %s``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
fwhmxy: (a float)
        Blur until the 2D (x,y)-plane FWHM reaches this value (in mm)
        argument: ``-FWHMxy %f``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
blurmaster: (an existing file name)
            The dataset whose smoothness controls the process.
            argument: ``-blurmaster %s``
automask: (a boolean)
          Create an automask from the input dataset.
          argument: ``-automask``
num_threads: (an integer (int or long), nipy default value: 1)
            set number of threads
environ: (a dictionary with keys which are a bytes or None or a 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)
      argument: ``-FWHM %f``
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.3.9 ClipLevel

[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: -1

[Optional]
mfrac: (a float)
        Use the number ff instead of 0.50 in the algorithm
        argument: ``-mfrac %s``, position: 2
doall: (a boolean)
        Apply the algorithm to each sub-brick separately.
        argument: ``-doall``, position: 3
        mutually_exclusive: g, r, a, d
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a 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: (a file name)
        Also compute a 'gradual' clip level as a function of voxel position,
        and output that to a dataset.
        argument: ``-grad %s``, position: 3
        mutually_exclusive: d, o, a, l, l
```

Outputs:

```
clip_val: (a float)
        output
```

51.3.10 DegreeCentrality

[Link to code](#)

Wraps the executable command 3dDegreeCentrality.

Performs degree centrality on a dataset using a given maskfile via 3dDegreeCentrality

For complete details, see the [3dDegreeCentrality Documentation](#).

Examples

```
>>> from nipyne.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
        argument: ``%s``, position: -1

[Optional]
mask: (an existing file name)
      mask file to mask input data
      argument: ``-mask %s``
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
num_threads: (an integer (int or long), nipyne default value: 1)
             set number of threads
oned_file: (a unicode string)
           output filepath to text dump of correlation matrix
           argument: ``-out1D %s``
environ: (a dictionary with keys which are a 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
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
           AFNI output filetype
thresh: (a float)
        threshold to exclude connections where corr <= thresh
        argument: ``-thresh %f``
autoclip: (a boolean)
          Clip off low-intensity regions in the dataset
          argument: ``-autoclip``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
polort: (an integer (int or long))
        argument: ``-polort %d``
automask: (a boolean)
          Mask the dataset to target brain-only voxels
          argument: ``-automask``
sparsity: (a float)
          only take the top percent of connections
          argument: ``-sparsity %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

51.3.11 Despiking

[Link to code](#)

Wraps the executable command 3dDespiking.

Removes 'spikes' from the 3D+time input dataset

For complete details, see the [3dDespiking Documentation](#).

Examples

```

>>> from nipyre.interfaces import afni
>>> despiking = afni.Despiking()
>>> despiking.inputs.in_file = 'functional.nii'
>>> despiking.cmdline
'3dDespiking -prefix functional_despiking functional.nii'
>>> res = despiking.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input file to 3dDespiking
    argument: ``%s``, position: -1

[Optional]
outtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
out_file: (a file name)
    output image file name
    argument: ``-prefix %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long), nipyre default value: 1)
    set number of threads
environ: (a dictionary with keys which are a 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

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.3.12 Detrend

[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.3.13 ECM

[Link to code](#)

Wraps the executable 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
         argument: ``%s``, position: -1

[Optional]
mask: (an existing file name)
      mask file to mask input data
      argument: ``-mask %s``
max_iter: (an integer (int or long))
          sets the maximum number of iterations to use in the power iteration;
          default = 1000
          argument: ``-max_iter %d``
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
       argument: ``-shift %f``
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
full: (a boolean)
      Full power method; enables thresholding; automatically selected if
      -thresh or -sparsity are set
      argument: ``-full``
num_threads: (an integer (int or long), nipy default value: 1)
            set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
scale: (a float)
      scale correlation coefficients in similarity matrix to after
      shifting, x >= 0.0; default = 1.0 for -full, 0.5 for -fecm
      argument: ``-scale %f``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
           AFNI output filetype
thresh: (a float)
       threshold to exclude connections where corr <= thresh
       argument: ``-thresh %f``
autoclip: (a boolean)
         Clip off low-intensity regions in the dataset
```

(continues on next page)

(continued from previous page)

```

        argument: ``-autoclip``
memory: (a float)
    Limit memory consumption on system by setting the amount of GB to
    limit the algorithm to; default = 2GB
    argument: ``-memory %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
polort: (an integer (int or long))
    argument: ``-polort %d``
automask: (a boolean)
    Mask the dataset to target brain-only voxels
    argument: ``-automask``
sparsity: (a float)
    only take the top percent of connections
    argument: ``-sparsity %f``
fecm: (a boolean)
    Fast centrality method; substantial speed increase but cannot
    accomodate thresholding; automatically selected if -thresh or
    -sparsity are not set
    argument: ``-fecm``
eps: (a float)
    sets the stopping criterion for the power iteration;  $12|v_{old} - v_{new}| < eps * |v_{old}|$ ; default = 0.001
    argument: ``-eps %f``

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.3.14 Fim[Link to code](#)

Wraps the executable 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
            argument: ``-ideal_file %s``, position: 2
in_file: (an existing file name)
          input file to 3dfim+
          argument: ``-input %s``, position: 1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
out_file: (a file name)
           output image file name
           argument: ``-bucket %s``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
out: (a unicode string)
     Flag to output the specified parameter
     argument: ``-out %s``, position: 4
num_threads: (an integer (int or long), nipy default value: 1)
              set number of threads
fim_thr: (a float)
          fim internal mask threshold value
          argument: ``-fim_thr %f``, 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
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.3.15 Fourier

[Link to code](#)

Wraps the executable 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
```

(continues on next page)

(continued from previous page)

```
'3dFourier -highpass 0.005000 -lowpass 0.100000 -prefix functional_fourier -
↳retrend functional.nii'
>>> res = fourier.run()
```

Inputs:

```
[Mandatory]
lowpass: (a float)
    lowpass
    argument: ``-lowpass %f``
in_file: (an existing file name)
    input file to 3dFourier
    argument: ``%s``, position: -1
highpass: (a float)
    highpass
    argument: ``-highpass %f``

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
out_file: (a file name)
    output image file name
    argument: ``-prefix %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
retrend: (a boolean)
    Any mean and linear trend are removed before filtering. This will
    restore the trend after filtering.
    argument: ``-retrend``
```

Outputs:

```
out_file: (an existing file name)
    output file
```

References:

None None

51.3.16 Hist[Link to code](#)

Wraps the executable 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
         argument: ``-input %s``, position: 1

[Optional]
mask: (an existing file name)
      matrix to align input file
      argument: ``-mask %s``
showhist: (a boolean, nipy default value: False)
           write a text visual histogram
           argument: ``-showhist``
out_show: (a file name)
           output image file name
           argument: ``> %s``, position: -1
min_value: (a float)
            minimum intensity value
            argument: ``-min %f``
out_file: (a file name)
           Write histogram to nlm file with this prefix
           argument: ``-prefix %s``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
max_value: (a float)
            maximum intensity value
            argument: ``-max %f``
bin_width: (a float)
            bin width
            argument: ``-binwidth %f``
nbin: (an integer (int or long))
       number of bins
       argument: ``-nbin %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
```

Outputs:

```
out_show: (a file name)
           output visual histogram
out_file: (an existing file name)
           output file
```

51.3.17 LFCD

[Link to code](#)

Wraps the executable 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
         argument: ``%s``, position: -1

[Optional]
mask: (an existing file name)
      mask file to mask input data
      argument: ``-mask %s``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
autoclip: (a boolean)
          Clip off low-intensity regions in the dataset
          argument: ``-autoclip``
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
thresh: (a float)
        threshold to exclude connections where corr <= thresh
        argument: ``-thresh %f``
automask: (a boolean)
          Mask the dataset to target brain-only voxels
          argument: ``-automask``
polort: (an integer (int or long))
        argument: ``-polort %d``
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.3.18 Maskave[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: -2

[Optional]
mask: (an existing file name)
      matrix to align input file
      argument: ``-mask %s``, position: 1
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
quiet: (a boolean)
       matrix to align input file
       argument: ``-quiet``, position: 2
out_file: (a file name)
          output image file name
          argument: ``> %s``, position: -1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
            set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.3.19 Means

[Link to code](#)

Wraps the executable command 3dMean.

Takes the voxel-by-voxel mean of all input datasets using 3dMean

For complete details, see the [3dMean Documentation](#).

Examples

```
>>> from nipyre.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 -prefix output.nii im1.nii im2.nii'
>>> res = means.run()
```

```
>>> from nipyre.interfaces import afni
>>> means = afni.Mean()
>>> means.inputs.in_file_a = 'im1.nii'
>>> means.inputs.out_file = 'output.nii'
>>> means.inputs.datum = 'short'
>>> means.cmdline
'3dMean -datum short -prefix output.nii im1.nii'
>>> res = means.run()
```

Inputs:

```
[Mandatory]
in_file_a: (an existing file name)
            input file to 3dMean
            argument: ``%s``, position: -2

[Optional]
count: (a boolean)
        compute count of non-zero voxels
        argument: ``-count``
datum: (a unicode string)
        Sets the data type of the output dataset
        argument: ``-datum %s``
summ: (a boolean)
        take sum, (not average)
        argument: ``-sum``
out_file: (a file name)
            output image file name
            argument: ``-prefix %s``
num_threads: (an integer (int or long), nipyre default value: 1)
              set number of threads
environ: (a dictionary with keys which are a 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
scale: (a unicode string)
        scaling of output
        argument: ``-%sscale``
non_zero: (a boolean)
```

(continues on next page)

(continued from previous page)

```

        use only non-zero values
        argument: ``-non_zero``
sqr: (a boolean)
    mean square instead of value
    argument: ``-sqr``
mask_inter: (a boolean)
    create intersection mask
    argument: ``-mask_inter``
in_file_b: (an existing file name)
    another input file to 3dMean
    argument: ``%s``, position: -1
std_dev: (a boolean)
    calculate std dev
    argument: ``-stdev``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
mask_union: (a boolean)
    create union mask
    argument: ``-mask_union``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.3.20 OutlierCount[Link to code](#)

Wraps the executable command 3dToutcount.

Calculates number of ‘outliers’ at each time point of a 3D+time dataset.

For complete details, see the [3dToutcount Documentation](#)**Examples**

```

>>> from nipy.interfaces import afni
>>> toutcount = afni.OutlierCount()
>>> toutcount.inputs.in_file = 'functional.nii'
>>> toutcount.cmdline
'3dToutcount -qthr 0.00100 functional.nii'
>>> res = toutcount.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input dataset
    argument: ``%s``, position: -2

[Optional]

```

(continues on next page)

(continued from previous page)

```

mask: (an existing file name)
    only count voxels within the given mask
    argument: ``-mask %s``
    mutually_exclusive: autoclip, automask
legendre: (a boolean, nipy default value: False)
    use Legendre polynomials
    argument: ``-legendre``
out_file: (a file name)
    capture standard output
outliers_file: (a file name)
    output image file name
    argument: ``-save %s``
save_outliers: (a boolean, nipy default value: False)
    enables out_file option
interval: (a boolean, nipy default value: False)
    write out the median + 3.5 MAD of outlier count with each timepoint
    argument: ``-range``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
qthr: (0.0 <= a floating point number <= 1.0, nipy default value:
    0.001)
    indicate a value for q to compute alpha
    argument: ``-qthr %.5f``
autoclip: (a boolean, nipy default value: False)
    clip off small voxels
    argument: ``-autoclip``
    mutually_exclusive: mask
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
automask: (a boolean, nipy default value: False)
    clip off small voxels
    argument: ``-automask``
    mutually_exclusive: mask
polort: (an integer (int or long))
    detrend each voxel timeseries with polynomials
    argument: ``-polort %d``
fraction: (a boolean, nipy default value: False)
    write out the fraction of masked voxels which are outliers at each
    timepoint
    argument: ``-fraction``

```

Outputs:

```

out_outliers: (an existing file name)
    output image file name
out_file: (a file name)
    capture standard output

```

51.3.21 QualityIndex[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: -2

[Optional]
mask: (an existing file name)
      compute correlation only across masked voxels
      argument: ``-mask %s``
      mutually_exclusive: autoclip, automask
clip: (a float)
      clip off values below
      argument: ``-clip %f``
autoclip: (a boolean, nipyne default value: False)
          clip off small voxels
          argument: ``-autoclip``
          mutually_exclusive: mask
out_file: (a file name)
          capture standard output
          argument: ``> %s``, position: -1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
spearman: (a boolean, nipyne default value: False)
          Quality index is 1 minus the Spearman (rank) correlation coefficient
          of each sub-brick with the median sub-brick. (default).
          argument: ``-spearman``
automask: (a boolean, nipyne default value: False)
          clip off small voxels
          argument: ``-automask``
          mutually_exclusive: mask
quadrant: (a boolean, nipyne default value: False)
          Similar to -spearman, but using 1 minus the quadrant correlation
          coefficient as the quality index.
          argument: ``-quadrant``
interval: (a boolean, nipyne default value: False)
          write out the median + 3.5 MAD of outlier count with each timepoint
          argument: ``-range``
environ: (a dictionary with keys which are a 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
```

Outputs:

```
out_file: (a file name)
          file containing the captured standard output
```

51.3.22 Qwarp

[Link to code](#)

Wraps the executable command 3dQwarp.

A version of 3dQwarp Allineate your images prior to passing them to this workflow.

For complete details, see the [3dQwarp Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> qwarp = afni.Qwarp()
>>> qwarp.inputs.in_file = 'sub-01_dir-LR_epi.nii.gz'
>>> qwarp.inputs.nopadWARP = True
>>> qwarp.inputs.base_file = 'sub-01_dir-RL_epi.nii.gz'
>>> qwarp.inputs.plusminus = True
>>> qwarp.cmdline
'3dQwarp -base sub-01_dir-RL_epi.nii.gz -source sub-01_dir-LR_epi.nii.gz -
↳nopadWARP -prefix sub-01_dir-LR_epi_QW -plusminus'
>>> res = qwarp.run()
```

```
>>> from nipy.interfaces import afni
>>> qwarp = afni.Qwarp()
>>> qwarp.inputs.in_file = 'structural.nii'
>>> qwarp.inputs.base_file = 'mni.nii'
>>> qwarp.inputs.resample = True
>>> qwarp.cmdline
'3dQwarp -base mni.nii -source structural.nii -prefix structural_QW -resample'
>>> res = qwarp.run()
```

```
>>> from nipy.interfaces import afni
>>> qwarp = afni.Qwarp()
>>> qwarp.inputs.in_file = 'structural.nii'
>>> qwarp.inputs.base_file = 'epi.nii'
>>> qwarp.inputs.out_file = 'anatSSQ.nii.gz'
>>> qwarp.inputs.resample = True
>>> qwarp.inputs.lpc = True
>>> qwarp.inputs.verb = True
>>> qwarp.inputs.iwarp = True
>>> qwarp.inputs.blur = [0,3]
>>> qwarp.cmdline
'3dQwarp -base epi.nii -blur 0.0 3.0 -source structural.nii -iwarp -prefix_
↳anatSSQ.nii.gz -resample -verb -lpc'
>>> res = qwarp.run()
```

```
>>> from nipy.interfaces import afni
>>> qwarp = afni.Qwarp()
>>> qwarp.inputs.in_file = 'structural.nii'
>>> qwarp.inputs.base_file = 'mni.nii'
>>> qwarp.inputs.duplo = True
>>> qwarp.inputs.blur = [0,3]
>>> qwarp.cmdline
```

(continues on next page)

(continued from previous page)

```
'3dQwarp -base mni.nii -blur 0.0 3.0 -duplo -source structural.nii -prefix_
↳structural_QW'
>>> res = qwarp.run()
```

```
>>> from nipy.interfaces import afni
>>> qwarp = afni.Qwarp()
>>> qwarp.inputs.in_file = 'structural.nii'
>>> qwarp.inputs.base_file = 'mni.nii'
>>> qwarp.inputs.duplo = True
>>> qwarp.inputs.minpatch = 25
>>> qwarp.inputs.blur = [0,3]
>>> qwarp.inputs.out_file = 'Q25'
>>> qwarp.cmdline
'3dQwarp -base mni.nii -blur 0.0 3.0 -duplo -source structural.nii -minpatch 25 -
↳prefix Q25'
>>> res = qwarp.run()
>>> qwarp2 = afni.Qwarp()
>>> qwarp2.inputs.in_file = 'structural.nii'
>>> qwarp2.inputs.base_file = 'mni.nii'
>>> qwarp2.inputs.blur = [0,2]
>>> qwarp2.inputs.out_file = 'Q11'
>>> qwarp2.inputs.inilev = 7
>>> qwarp2.inputs.iniwarp = ['Q25_warp+tlrc.HEAD']
>>> qwarp2.cmdline
'3dQwarp -base mni.nii -blur 0.0 2.0 -source structural.nii -inilev 7 -iniwarp_
↳Q25_warp+tlrc.HEAD -prefix Q11'
>>> res2 = qwarp2.run()
>>> res2 = qwarp2.run()
>>> qwarp3 = afni.Qwarp()
>>> qwarp3.inputs.in_file = 'structural.nii'
>>> qwarp3.inputs.base_file = 'mni.nii'
>>> qwarp3.inputs.allineate = True
>>> qwarp3.inputs.allineate_opts = '-cose lpa -verb'
>>> qwarp3.cmdline
'3dQwarp -allineate -allineate_opts '-cose lpa -verb' -base mni.nii -source_
↳structural.nii -prefix structural_QW'
>>> res3 = qwarp3.run()
```

Inputs:

[Mandatory]

in_file: (an existing file name)
Source image (opposite phase encoding direction than base image).
argument: ``-source %s``

base_file: (an existing file name)
Base image (opposite phase encoding direction than source image).
argument: ``-base %s``

[Optional]

allineate: (a boolean)
This option will make 3dQwarp run 3dAllineate first, to align the source dataset to the base with an affine transformation. It will then use that alignment as a starting point for the nonlinear warping.
argument: ``-allineate``

Qonly: (a boolean)
Use Hermite quintic polynomials at all levels.* Very slow (about 4

(continues on next page)

(continued from previous page)

```

times longer). Also experimental.* Will produce a (discrete
representation of a) C2 warp.
argument: ``-Qonly``
out_weight_file: (a file name)
Write the weight volume to disk as a dataset
argument: ``-wtprefix %s``
expad: (an integer (int or long))
This option instructs the program to pad the warp by an extra 'EE'
voxels (and then 3dQwarp starts optimizing it).* This option is
seldom needed, but can be useful if you might later concatenate the
nonlinear warp -- via 3dNwarpCat -- with an affine transformation
that contains a large shift. Under that circumstance, the nonlinear
warp might be shifted partially outside its original grid, so
expanding that grid can avoid this problem.* Note that this option
perforce turns off '-nopadWARP'.
argument: ``-expad %d``
mutually_exclusive: nopadWARP
penfac: (a float)
Use this value to weight the penalty. The default value is 1. Larger
values mean the penalty counts more, reducing grid
distortions, insha'Allah; '-nopenalty' is the same as '-penfac 0'.
-->* [23 Sep 2013] -- Zhark increased the default value of the
penalty by a factor of 5, and also made it get progressively larger
with each level of refinement. Thus, warping results will vary from
earlier instances of 3dQwarp. * The progressive increase in the
penalty at higher levels means that the 'cost function' can actually
look like the alignment is getting worse when the levels change. *
IF you wish to turn off this progression, for whatever reason (e.g.,
to keep compatibility with older results), use the option
'-penold'. To be completely compatible with the older 3dQwarp, you'll
also have to use '-penfac 0.2'.
argument: ``-penfac %f``
weight: (an existing file name)
Instead of computing the weight from the base dataset, directly input
the weight volume from dataset 'www'.* Useful if you know what over
parts of the base image you want to emphasize or de-emphasize the
matching functional.
argument: ``-weight %s``
inilev: (an integer (int or long))
The initial refinement 'level' at which to start.* Usually used with
-iniwarp; CANNOT be used with -duplo.* The combination of -inilev
and -iniwarp lets you take the results of a previous 3dQwarp run and
refine them further: Note that the source dataset in the second run
is the SAME as in the first run. If you don't see why this is
necessary, then you probably need to seek help from an AFNI guru.
argument: ``-inilev %d``
mutually_exclusive: duplo
workhard: (a boolean)
Iterate more times, which can help when the volumes are hard to align
at all, or when you hope to get a more precise alignment.* Slows the
program down (possibly a lot), of course.* When you combine
'-workhard' with '-duplo', only the full size volumes get the extra
iterations.* For finer control over which refinement levels work
hard, you can use this option in the form (for example)
-workhard:4:7 which implies the extra iterations will be done at
levels 4, 5, 6, and 7, but not otherwise.* You can also use
'-superhard' to iterate even more, but this extra option will REALLY

```

(continues on next page)

(continued from previous page)

```

slow things down.-->*> Under most circumstances, you should not need
to use either -workhard or -superhard.-->*> The fastest way to
register to a template image is via the -duplo option, and without
the -workhard or -superhard options.-->*> If you use this option in
the form '-Workhard' (first letter in upper case), then the second
iteration at each level is done with quintic polynomial warps.
argument: ``-workhard``
mutually_exclusive: boxopt, ballopt
noZdis: (a boolean)
Warp will not displace in z direction
argument: ``-noZdis``
nmi: (a boolean)
Normalized Mutual Information: a matching function for the
adventurous This option has NOT been extensively tested for
usefulness and should be considered experimental at this
infundibulum.
argument: ``-nmi``
mutually_exclusive: nmi, hel, lpc, lpa, pear
minpatch: (an integer (int or long))
* The value of mm should be an odd integer.* The default value of mm
is 25.* For more accurate results than mm=25, try 19 or 13.* The
smallest allowed patch size is 5.* You may want to stop at a larger
patch size (say 7 or 9) and use the -Qfinal option to run that final
level with quintic warps, which might run faster and provide the same
degree of warp detail.* Trying to make two different brain volumes
match in fine detail is usually a waste of time, especially in
humans. There is too much variability in anatomy to match gyrus to
gyrus accurately. For this reason, the default minimum patch size is
25 voxels. Using a smaller '-minpatch' might try to force the warp
to match features that do not match, and the result can be
useless image distortions -- another reason to LOOK AT THE RESULTS.
argument: ``-minpatch %d``
duplo: (a boolean)
Start off with 1/2 scale versions of the volumes, for getting a
speedy coarse first alignment.* Then scale back up to register the
full volumes. The goal is greater speed, and it seems to help
this positively piggish program to be more expeditious.* However,
accuracy is somewhat lower with '-duplo', for reasons that currently
elude Zhark; for this reason, the Emperor does not usually use
'-duplo'.
argument: ``-duplo``
mutually_exclusive: gridlist, maxlev, inilev, iniwarp, plusminus,
allsave
iwarp: (a boolean)
Do compute and save the _WARPINV file.
argument: ``-iwarp``
mutually_exclusive: plusminus
noneg: (a boolean)
Replace negative values in either input volume with 0.* If there ARE
negative input values, and you do NOT use -noneg, then strict Pearson
correlation will be used, since the 'clipped' method only is
implemented for non-negative volumes.* '-noneg' is not the default,
since there might be situations where you want to align datasets with
positive and negative values mixed.* But, in many cases, the
negative values in a dataset are just the result of interpolation
artifacts (or other peculiarities), and so they should be ignored.
That is what '-noneg' is for.

```

(continues on next page)

(continued from previous page)

```

        argument: ``-noneg``
gridlist: (an existing file name)
    This option provides an alternate way to specify the patchgrid sizes
    used in the warp optimization process. 'gl' is a 1D file with a list
    of patches to use -- in most cases, you will want to use it in the
    following form: -gridlist '1D: 0 151 101 75 51'* Here, a 0 patch size
    means the global domain. Patch sizes otherwise should be odd integers
    >= 5.* If you use the '0' patch size again after the first
    position, you will actually get an iteration at the size of
    the default patch level 1, where the patch sizes are 75% of the volume
    dimension. There is no way to force the program to literally repeat
    the same step of lev=0.* You cannot use -gridlist with -duplo
    or -plusminus!
    argument: ``-gridlist %s``
mutually_exclusive: duplo, plusminus
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
hel: (a boolean)
    Hellinger distance: a matching function for the adventurous. This
    option has NOT been extensively tested for usefulness and should be
    considered experimental at this infundibulum.
    argument: ``-hel``
mutually_exclusive: nmi, mi, lpc, lpa, pear
pear: (a boolean)
    Use strict Pearson correlation for matching.* Not usually
    recommended, since the 'clipped Pearson' method used by default will
    reduce the impact of outlier values.
    argument: ``-pear``
allineate_opts: (a unicode string)
    add extra options to the 3dAllineate command to be run by 3dQwarp.
    argument: ``-allineate_opts %s``
requires: allineate
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
ballopt: (a boolean)
    Normally, the incremental warp parameters are optimized inside a
    rectangular 'box' (24 dimensional for cubic patches, 81 for quintic
    patches), whose limits define the amount of distortion allowed at
    each step. Using '-ballopt' switches these limits to be applied to a
    'ball' (interior of a hypersphere), which can allow for larger
    incremental displacements. Use this option if you think things need
    to be able to move farther.
    argument: ``-ballopt``
mutually_exclusive: workhard, boxopt
maxlev: (an integer (int or long))
    The initial refinement 'level' at which to start.* Usually used with
    -iniwarp; CANNOT be used with -duplo.* The combination of -inilev
    and -iniwarp lets you take the results of a previous 3dQwarp run and
    refine them further: Note that the source dataset in the second run
    is the SAME as in the first run. If you don't see why this is
    necessary, then you probably need to seek help from an AFNI guru.
    argument: ``-maxlev %d``, position: -1
mutually_exclusive: duplo
noYdis: (a boolean)
    Warp will not displace in y direction
    argument: ``-noYdis``

```

(continues on next page)

(continued from previous page)

```

baxopt: (a boolean)
    Use the 'box' optimization limits instead of the 'ball'[this is the
    default at present].* Note that if '-workhard' is used, then ball
    and box optimization are alternated in the different iterations at
    each level, so these two options have no effect in that case.
    argument: ``-boxopt``
    mutually_exclusive: workhard, ballopt
iniwarp: (a list of items which are an existing file name)
    A dataset with an initial nonlinear warp to use.* If this option is
    not used, the initial warp is the identity.* You can specify a
    concatenation of warps (in quotes) here, as in program 3dNwarpApply.* As
    a special case, if you just input an affine matrix in a .1D file,
    that will work also -- it is treated as giving the initial warp via
    the string "IDENT(base_dataset) matrix_file.aff12.1D".* You CANNOT
    use this option with -duplo !!* -iniwarp is usually used with
    -inilev to re-start 3dQwarp from a previous stopping point.
    argument: ``-iniwarp %s``
    mutually_exclusive: duplo
plusminus: (a boolean)
    Normally, the warp displacements  $dis(x)$  are defined to match  $base(x)$ 
    to  $source(x+dis(x))$ . With this option, the match is between
     $base(x-dis(x))$  and  $source(x+dis(x))$  -- the two images 'meet in the
    middle'.* One goal is to mimic the warping done to MRI EPI data
    by field inhomogeneities, when registering between a 'blip up' and a
    'blip down' down volume, which will have opposed distortions.*
    Define  $Wp(x) = x+dis(x)$  and  $Wm(x) = x-dis(x)$ . Then since  $base(Wm(x))$ 
    matches  $source(Wp(x))$ , by substituting  $INV(Wm(x))$  wherever we see  $x$ ,
    we have  $base(x)$  matches  $source(Wp(INV(Wm(x))))$ ; that is, the warp
     $V(x)$  that one would get from the 'usual' way of running 3dQwarp is
     $V(x) = Wp(INV(Wm(x)))$ .* Conversely, we can calculate  $Wp(x)$  in terms
    of  $V(x)$  as follows: If  $V(x) = x + dv(x)$ , define  $Vh(x) = x +$ 
 $dv(x)/2$ ; then  $Wp(x) = V(INV(Vh(x)))$ * With the above formulas, it is
    possible to compute  $Wp(x)$  from  $V(x)$  and vice-versa, using program
    3dNwarpCalc. The requisite commands are left as an exercise for the
    aspiring AFNI Jedi Master.* You can use the semi-secret '-pmBASE'
    option to get the  $V(x)$  warp and the source dataset warped to base
    space, in addition to the  $Wp(x)$  '_PLUS' and  $Wm(x)$  '_MINUS'
    warps.-->* Alas: -plusminus does not work with -duplo or -allineate
    :-( * However, you can use -iniwarp with -plusminus :-( -->* The
    outputs have _PLUS (from the source dataset) and _MINUS (from the
    base dataset) in their filenames, in addition to the prefix. The
    -iwarp option, if present, will be ignored.
    argument: ``-plusminus``
    mutually_exclusive: duplo, allsave, iwarp
blur: (a list of from 1 to 2 items which are a float)
    Gaussian blur the input images by 'bb' (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
    for 'bb', and then the first one is applied to the base volume, the
    second to the source volume.-->* e.g., '-blur 0 3' to skip blurring
    the base image (if the base is a blurry template, for example).* 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.* If the base is a template
    volume that is already blurry, you probably don't want to blur it
    again, but blurring the source volume a little is probably a good
    idea, to help the program avoid trying to match tiny features.* Note

```

(continues on next page)

(continued from previous page)

```

that -duplo will blur the volumes some extraamount for the initial
small-scale warping, to makethat phase of the program converge more
rapidly.
argument: ``-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
noXdis: (a boolean)
        Warp will not displace in x directoin
argument: ``-noXdis``
lpc: (a boolean)
        Local Pearson minimization (i.e., EPI-T1 registration)This option
        has not be extensively testedIf you use '-lpc', then '-maxlev 0' is
        automatically set.If you want to go to more refined levels, you can
        set '-maxlev'This should be set up to have lpc as the second to last
        argumentand maxlev as the second to last argument, as needed by
        AFNIUsing maxlev > 1 is not recommended for EPI-T1 alignment.
argument: ``-lpc``, position: -2
mutually_exclusive: nmi, mi, hel, lpa, pear
overwrite: (a boolean)
        Overwrite outputs
argument: ``-overwrite``
mi: (a boolean)
        Mutual Information: a matching function for the adventurousThis
        option has NOT be extensively tested for usefullnessand should be
        considered experimental at this infundibulum.
argument: ``-mi``
mutually_exclusive: mi, hel, lpc, lpa, pear
nopad: (a boolean)
        Do NOT use zero-padding on the 3D base and source images.[Default ==
        zero-pad, if needed]* The underlying model for deformations goes to
        zero at theedge of the volume being warped. However, if there
        issignificant data near an edge of the volume, then it won'tget
        displaced much, and so the results might not be good.* Zero padding
        is designed as a way to work around this potentialproblem. You
        should NOT need the '-nopad' option for anyreason that Zhark can
        think of, but it is here to be symmetricalwith 3dAllineate.* Note
        that the output (warped from source) dataset will be on thebase
        dataset grid whether or not zero-padding is allowed. However,unless
        you use the following option, allowing zero-padding (i.e.,the
        default operation) will make the output WARP dataset(s) beon a
        larger grid (also see '-expad' below).
argument: ``-nopad``
quiet: (a boolean)
        Cut out most of the fun fun fun progress messages :-(
argument: ``-quiet``
mutually_exclusive: verb
wmask: (a tuple of the form: (an existing file name, a float))
        -wmask ws fSimilar to '-wball', but here, you provide a dataset
        'ws'that indicates where to increase the weight.* The 'ws' dataset
        must be on the same 3D grid as the base dataset.* 'ws' is treated as
        a mask -- it only matters where itis nonzero -- otherwise, the
        values inside are not used.* After 'ws' comes the factor 'f' by
        which to increase theautomatically computed weight. Where 'ws' is
        nonzero,the weighting will be multiplied by (1+f).* As with
        '-wball', the factor 'f' should be between 1 and 100.* You cannot

```

(continues on next page)

(continued from previous page)

```

    use '-wball' and '-wmask' together!
    argument: ``-wpass %s %f``
emask: (an existing file name)
    Here, 'ee' is a dataset to specify a mask of voxelsto EXCLUDE from
    the analysis -- all voxels in 'ee'that are NONZERO will not be used
    in the alignment.* The base image always automasked -- the emask
    isextra, to indicate voxels you definitely DON'T wantincluded in the
    matching process, even if they areinside the brain.
    argument: ``-emask %s``
nowarp: (a boolean)
    Do not save the _WARP file.
    argument: ``-nowarp``
Qfinal: (a boolean)
    At the finest patch size (the final level), use Hermitequintic
    polynomials for the warp instead of cubic polynomials.* In a 3D
    'patch', there are 2x2x2x3=24 cubic polynomial basisfunction
    parameters over which to optimize (2 polynomialsdependent on each of
    the x,y,z directions, and 3 differentdirections of displacement).*
    There are 3x3x3x3=81 quintic polynomial parameters per patch.* With
    -Qfinal, the final level will have more detail inthe allowed warps,
    at the cost of yet more CPU time.* However, no patch below 7x7x7 in
    size will be done with quinticpolynomials.* This option is also not
    usually needed, and is experimental.
    argument: ``-Qfinal``
resample: (a boolean)
    This option simply resamples the source dataset to match thebase
    dataset grid. You can use this if the two datasetsoverlap well (as
    seen in the AFNI GUI), but are not on thesame 3D grid.* If they
    don't overlap well, allineate them first* The reampling here is done
    with the'wsinc5' method, which has very little blurring artifact.*
    If the base and source datasets ARE on the same 3D grid,then the
    -resample option will be ignored.* You CAN use -resample with these
    3dQwarp options:-plusminus -inilev -iniwarp -duplo
    argument: ``-resample``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
pblur: (a list of from 1 to 2 items which are a float)
    Use progressive blurring; that is, for larger patch sizes,the amount
    of blurring is larger. The general idea is toavoid trying to match
    finer details when the patch sizeand incremental warps are coarse.
    When '-blur' is usedas well, it sets a minimum amount of blurring
    that willbe used. [06 Aug 2014 -- '-pblur' may become the default
    someday].* You can optionally give the fraction of the patch size
    thatis used for the progressive blur by providing a value between0
    and 0.25 after '-pblur'. If you provide TWO values, thethe first
    fraction is used for progressively blurring thebase image and the
    second for the source image. The defaultparameters when just
    '-pblur' is given is the same as givingthe options as '-pblur 0.09
    0.09'.* '-pblur' is useful when trying to match 2 volumes with
    highamounts of detail; e.g, warping one subject's brain image
    tomatch another's, or trying to warp to match a detailed template.*
    Note that using negative values with '-blur' means that
    theprogressive blurring will be done with median filters, ratherthan
    Gaussian linear blurring.-->>*** The combination of the -allineate
    and -pblur options will makethe results of using 3dQwarp to align to
    a template somewhatless sensitive to initial head position and
    scaling.

```

(continues on next page)

(continued from previous page)

```

    argument: ``-pblur %s``
verb: (a boolean)
    more detailed description of the process
    argument: ``-verb``
    mutually_exclusive: quiet
nopenalty: (a boolean)
    Replace negative values in either input volume with 0.* If there ARE
    negative input values, and you do NOT use -noneg, then strict Pearson
    correlation will be used, since the 'clipped' method only is
    implemented for non-negative volumes.* '-noneg' is not the default,
    since there might be situations where you want to align datasets with
    positive and negative values mixed.* But, in many cases, the
    negative values in a dataset are just the result of interpolation
    artifacts (or other peculiarities), and so they should be ignored.
    That is what '-noneg' is for.
    argument: ``-nopenalty``
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.
    argument: ``-noweight``
allsave: (a boolean)
    This option lets you save the output warps from each level of the
    refinement process. Mostly used for experimenting.* Cannot be used
    with -nopadWARP, -duplo, or -plusminus.* Will only save all the
    outputs if the program terminates normally -- if it crashes, or
    freezes, then all these warps are lost.
    argument: ``-allsave``
    mutually_exclusive: nopadWARP, duplo, plusminus
lpa: (a boolean)
    Local Pearson maximization This option has not been extensively tested
    argument: ``-lpa``
    mutually_exclusive: nmi, mi, lpc, hel, pear
nopadWARP: (a boolean)
    If for some reason you require the warp volume to match the base
    volume, then use this option to have the output WARP dataset(s)
    truncated.
    argument: ``-nopadWARP``
    mutually_exclusive: allsave, expad
out_file: (a file name)
    out_file ppp Sets the prefix for the output datasets.* The source
    dataset is warped to match the base and gets prefix 'ppp'. (Except if
    '-plusminus' is used.)* The final interpolation to this output
    dataset is done using the 'wsinc5' method. See the output of
    3dAllineate -HELP (in the "Modifying '-final wsinc5'" section) for the
    lengthy technical details.* The 3D warp used is saved in a dataset
    with prefix 'ppp_WARP' -- this dataset can be used with 3dNwarpApply
    and 3dNwarpCat, for example.* To be clear, this is the warp from
    source dataset coordinates to base dataset coordinates, where the
    values at each base grid point are the xyz displacements needed to
    move that grid point's xyz values to the corresponding xyz values in
    the source dataset: base( (x,y,z) + WARP(x,y,z) ) matches
    source(x,y,z) Another way to think of this warp is that it 'pulls'
    values back from source space to base space.* 3dNwarpApply would use
    'ppp_WARP' to transform datasets aligned with the source dataset to
    be aligned with the base dataset.** If you do NOT want this warp
    saved, use the option '-nowarp'.-->> (However, this warp is usually

```

(continues on next page)

(continued from previous page)

```

the most valuable possible output!)* If you want to calculate and
save the inverse 3D warp, use the option '-iwarp'. This inverse warp
will then be saved in a dataset with prefix 'ppp_WARPINV'.* This
inverse warp could be used to transform data from basespace to
source space, if you need to do such an operation.* You can easily
compute the inverse later, say by a command like 3dNwarpCat -prefix
Z_WARPINV 'INV(Z_WARP+tlrc)' or the inverse can be computed as needed
in 3dNwarpApply, like 3dNwarpApply -nwarp 'INV(Z_WARP+tlrc)' -source
Dataset.nii ...
argument: ``-prefix %s``
wball: (a list of from 5 to 5 items which are an integer (int or
long))
-wball x y z r f Enhance automatic weight from '-useweight' by a
factor of 1+f*Gaussian(FWHM=r) centered in the base image at DICOM
coordinates (x,y,z) and with radius 'r'. The goal of this option is
to try and make the alignment better in a specific part of the
brain.* Example: -wball 0 14 6 30 40 to emphasize the thalamic area
(in MNI/Talairach space).* The 'r' parameter must be positive!* The
'f' parameter must be between 1 and 100 (inclusive).* '-wball' does
nothing if you input your own weight with the '-weight' option.*
'-wball' does change the binary weight created by the '-noweight'
option.* You can only use '-wball' once in a run of 3dQwarp.*** The
effect of '-wball' is not dramatic. The example above makes the
average brain image across a collection of subjects a little sharper
in the thalamic area, which might have some small value. If you care
enough about alignment to use '-wball', then you should examine
the results from 3dQwarp for each subject, to see if the alignments
are good enough for your purposes.
argument: ``-wball %s``

```

Outputs:

```

warped_source: (a file name)
    Warped source file. If plusminus is used, this is the
    undistorted source file.
warped_base: (a file name)
    Undistorted base file.
base_warp: (a file name)
    Displacement in mm for the base image. If plusminus is used, this is
    the field susceptibility correction warp (in 'mm') for base image.
    This is only output if plusminus or iwarp options are passed
weights: (a file name)
    Auto-computed weight volume.
source_warp: (a file name)
    Displacement in mm for the source image. If plusminus is used this is
    the field susceptibility correction warp (in 'mm') for source image.

```

References:

None None

51.3.23 QwarpPlusMinus[Link to code](#)

Wraps the executable command 3dQwarp.

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.in_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 -
  ↪source sub-01_dir-LR_epi.nii.gz -nopadWARP'
>>> res = warp.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    Source image (opposite phase encoding direction than base image).
    argument: ``-source %s``
base_file: (an existing file name)
    Base image (opposite phase encoding direction than source image).
    argument: ``-base %s``

[Optional]
nopenalty: (a boolean)
    Replace negative values in either input volume with 0.* If there ARE
    negative input values, and you do NOT use -noneg, then strict Pearson
    correlation will be used, since the 'clipped' method only is
    implemented for non-negative volumes.* '-noneg' is not the default,
    since there might be situations where you want to align datasets with
    positive and negative values mixed.* But, in many cases, the
    negative values in a dataset are just the result of interpolation
    artifacts (or other peculiarities), and so they should be ignored.
    That is what '-noneg' is for.
    argument: ``-nopenalty``
allineate: (a boolean)
    This option will make 3dQwarp run 3dAllineate first, to align the
    source dataset to the base with an affine transformation. It will
    then use that alignment as a starting point for the nonlinear
    warping.
    argument: ``-allineate``
Qonly: (a boolean)
    Use Hermite quintic polynomials at all levels.* Very slow (about 4
    times longer). Also experimental.* Will produce a (discrete
    representation of a) C2 warp.
    argument: ``-Qonly``
out_weight_file: (a file name)
    Write the weight volume to disk as a dataset
    argument: ``-wtprefix %s``
expad: (an integer (int or long))
    This option instructs the program to pad the warp by an extra 'EE'
    voxels (and then 3dQwarp starts optimizing it).* This option is
    seldom needed, but can be useful if you might later concatenate the
    nonlinear warp -- via 3dNwarpCat -- with an affine transformation
    that contains a large shift. Under that circumstance, the nonlinear
    warp might be shifted partially outside its original grid, so
    expanding that grid can avoid this problem.* Note that this option
```

(continues on next page)

(continued from previous page)

```

perforce turns off '-nopadWARP'.
argument: ``-expad %d``
mutually_exclusive: nopadWARP
penfac: (a float)
    Use this value to weight the penalty. The default value is 1. Larger
    values mean the penalty counts more, reducing grid
    distortions, insha'Allah; '-nopenalty' is the same as '-penfac 0'.
    -->* [23 Sep 2013] -- Zhark increased the default value of the
    penalty by a factor of 5, and also made it get progressively larger
    with each level of refinement. Thus, warping results will vary from
    earlier instances of 3dQwarp. * The progressive increase in the
    penalty at higher levels means that the 'cost function' can actually
    look like the alignment is getting worse when the levels change. *
    IF you wish to turn off this progression, for whatever reason (e.g.,
    to keep compatibility with older results), use the option
    '-penold'. To be completely compatible with the older 3dQwarp, you'll
    also have to use '-penfac 0.2'.
    argument: ``-penfac %f``
inilev: (an integer (int or long))
    The initial refinement 'level' at which to start. * Usually used with
    -iniwarp; CANNOT be used with -duplo. * The combination of -inilev
    and -iniwarp lets you take the results of a previous 3dQwarp run and
    refine them further: Note that the source dataset in the second run
    is the SAME as in the first run. If you don't see why this is
    necessary, then you probably need to seek help from an AFNI guru.
    argument: ``-inilev %d``
    mutually_exclusive: duplo
workhard: (a boolean)
    Iterate more times, which can help when the volumes are hard to align
    at all, or when you hope to get a more precise alignment. * Slows the
    program down (possibly a lot), of course. * When you combine
    '-workhard' with '-duplo', only the full size volumes get the extra
    iterations. * For finer control over which refinement levels work
    hard, you can use this option in the form (for example)
    -workhard:4:7 which implies the extra iterations will be done at
    levels 4, 5, 6, and 7, but not otherwise. * You can also use
    '-superhard' to iterate even more, but this extra option will REALLY
    slow things down. -->* Under most circumstances, you should not need
    to use either -workhard or -superhard. -->* The fastest way to
    register to a template image is via the -duplo option, and without
    the -workhard or -superhard options. -->* If you use this option in
    the form '-Workhard' (first letter in upper case), then the second
    iteration at each level is done with quintic polynomial warps.
    argument: ``-workhard``
    mutually_exclusive: boxopt, ballopt
noZdis: (a boolean)
    Warp will not displace in z direction
    argument: ``-noZdis``
nmi: (a boolean)
    Normalized Mutual Information: a matching function for the
    adventurous. This option has NOT been extensively tested for
    usefulness and should be considered experimental at this
    infundibulum.
    argument: ``-nmi``
    mutually_exclusive: nmi, hel, lpc, lpa, pear
minpatch: (an integer (int or long))
    * The value of mm should be an odd integer. * The default value of mm

```

(continues on next page)

(continued from previous page)

```

is 25.* For more accurate results than mm=25, try 19 or 13.* The
smallest allowed patch size is 5.* You may want stop at a larger
patch size (say 7 or 9) and use the -Qfinal option to run that final
level with quintic warps, which might run faster and provide the same
degree of warp detail.* Trying to make two different brain volumes
match in fine detail is usually a waste of time, especially in
humans. There is too much variability in anatomy to match gyri to
gyri accurately. For this reason, the default minimum patch size is
25 voxels. Using a smaller '-minpatch' might try to force the warp
to match features that do not match, and the result can be
useless image distortions -- another reason to LOOK AT THE RESULTS.
argument: ``-minpatch %d``

duplo: (a boolean)
Start off with 1/2 scale versions of the volumes, for getting a
speedy coarse first alignment.* Then scale back up to register the
full volumes. The goal is greater speed, and it seems to help
this positively piggyback program to be more expeditious.* However,
accuracy is somewhat lower with '-duplo', for reasons that currently
elude Zhark; for this reason, the Emperor does not usually use
'-duplo'.
argument: ``-duplo``
mutually_exclusive: gridlist, maxlev, inilev, iniwarp, plusminus,
    allsave

iwarp: (a boolean)
Do compute and save the _WARPINV file.
argument: ``-iwarp``
mutually_exclusive: plusminus

noneg: (a boolean)
Replace negative values in either input volume with 0.* If there ARE
negative input values, and you do NOT use -noneg, then strict Pearson
correlation will be used, since the 'clipped' method only is
implemented for non-negative volumes.* '-noneg' is not the default,
since there might be situations where you want to align datasets with
positive and negative values mixed.* But, in many cases, the
negative values in a dataset are just the result of interpolation
artifacts (or other peculiarities), and so they should be ignored.
That is what '-noneg' is for.
argument: ``-noneg``

gridlist: (an existing file name)
This option provides an alternate way to specify the patch grid sizes
used in the warp optimization process. 'gl' is a 1D file with a list
of patches to use -- in most cases, you will want to use it in the
following form: -gridlist '1D: 0 151 101 75 51'* Here, a 0 patch size
means the global domain. Patch sizes otherwise should be odd integers
>= 5.* If you use the '0' patch size again after the first
position, you will actually get an iteration at the size of
the default patch level 1, where the patch sizes are 75% of the volume
dimension. There is no way to force the program to literally repeat
the same generic step of lev=0.* You cannot use -gridlist with -duplo
or -plusminus!
argument: ``-gridlist %s``
mutually_exclusive: duplo, plusminus

outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
AFNI output filetype

hel: (a boolean)
Hellinger distance: a matching function for the adventurous. This
option has NOT been extensively tested for usefulness and should be

```

(continues on next page)

(continued from previous page)

```

    considered experimental at this infundibulum.
    argument: ``-hel``
    mutually_exclusive: nmi, mi, lpc, lpa, pear
pear: (a boolean)
    Use strict Pearson correlation for matching.* Not usually
    recommended, since the 'clipped Pearson' method used by default will
    reduce the impact of outlier values.
    argument: ``-pear``
allineate_opts: (a unicode string)
    add extra options to the 3dAllineate command to be run by 3dQwarp.
    argument: ``-allineate_opts %s``
    requires: allineate
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
overwrite: (a boolean)
    Overwrite outputs
    argument: ``-overwrite``
ballopt: (a boolean)
    Normally, the incremental warp parameters are optimized inside a
    rectangular 'box' (24 dimensional for cubic patches, 81 for quintic
    patches), whose limits define the amount of distortion allowed at
    each step. Using '-ballopt' switches these limits to be applied to a
    'ball' (interior of a hypersphere), which can allow for larger
    incremental displacements. Use this option if you think things need
    to be able to move farther.
    argument: ``-ballopt``
    mutually_exclusive: workhard, boxopt
maxlev: (an integer (int or long))
    The initial refinement 'level' at which to start.* Usually used with
    -iniwarp; CANNOT be used with -duplo.* The combination of -inilev
    and -iniwarp lets you take the results of a previous 3dQwarp run and
    refine them further: Note that the source dataset in the second run
    is the SAME as in the first run. If you don't see why this is
    necessary, then you probably need to seek help from an AFNI guru.
    argument: ``-maxlev %d``, position: -1
    mutually_exclusive: duplo
noYdis: (a boolean)
    Warp will not displace in y direction
    argument: ``-noYdis``
boxopt: (a boolean)
    Use the 'box' optimization limits instead of the 'ball' [this is the
    default at present].* Note that if '-workhard' is used, then ball
    and box optimization are alternated in the different iterations at
    each level, so these two options have no effect in that case.
    argument: ``-boxopt``
    mutually_exclusive: workhard, ballopt
iniwarp: (a list of items which are an existing file name)
    A dataset with an initial nonlinear warp to use.* If this option is
    not used, the initial warp is the identity.* You can specify a
    catenation of warps (in quotes) here, as in program 3dNwarpApply.* As
    a special case, if you just input an affine matrix in a .1D file,
    that will work also -- it is treated as giving the initial warp via
    the string "IDENT(base_dataset) matrix_file.aff12.1D".* You CANNOT
    use this option with -duplo !!* -iniwarp is usually used with
    -inilev to re-start 3dQwarp from a previous stopping point.
    argument: ``-iniwarp %s``

```

(continues on next page)

(continued from previous page)

```

    mutually_exclusive: duplo
plusminus: (a boolean, nipy default value: True)
    Normally, the warp displacements dis(x) are defined to matchbase(x)
    to source(x+dis(x)). With this option, the match is between
    base(x-dis(x)) and source(x+dis(x)) -- the two images 'meet in the
    middle'. For more info, view Qwarp` interface
    argument: ``-plusminus``, position: 1
    mutually_exclusive: duplo, allsave, iwarp
blur: (a list of from 1 to 2 items which are a float)
    Gaussian blur the input images by 'bb' (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
    for 'bb', and then the first one is applied to the base volume, the
    second to the source volume. --> * e.g., '-blur 0 3' to skip blurring
    the base image (if the base is a blurry template, for example). * 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. * If the base is a template
    volume that is already blurry, you probably don't want to blur it
    again, but blurring the source volume a little is probably a good
    idea, to help the program avoid trying to match tiny features. * Note
    that -duplo will blur the volumes some extra amount for the initial
    small-scale warping, to make that phase of the program converge more
    rapidly.
    argument: ``-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
source_file: (an existing file name)
    Source image (opposite phase encoding direction than base image)
    argument: ``-source %s``
lpc: (a boolean)
    Local Pearson minimization (i.e., EPI-T1 registration) This option
    has not been extensively tested. If you use '-lpc', then '-maxlev 0' is
    automatically set. If you want to go to more refined levels, you can
    set '-maxlev'. This should be set up to have lpc as the second to last
    argument and maxlev as the second to last argument, as needed by
    AFNI. Using maxlev > 1 is not recommended for EPI-T1 alignment.
    argument: ``-lpc``, position: -2
    mutually_exclusive: nmi, mi, hel, lpa, pear
noXdis: (a boolean)
    Warp will not displace in x direction
    argument: ``-noXdis``
nopad: (a boolean)
    Do NOT use zero-padding on the 3D base and source images. [Default ==
    zero-pad, if needed] * The underlying model for deformations goes to
    zero at the edge of the volume being warped. However, if there is
    insignificant data near an edge of the volume, then it won't get
    displaced much, and so the results might not be good. * Zero padding
    is designed as a way to work around this potential problem. You
    should NOT need the '-nopad' option for any reason that Zhark can
    think of, but it is here to be symmetrical with 3dAllineate. * Note
    that the output (warped from source) dataset will be on the base
    dataset grid whether or not zero-padding is allowed. However, unless
    you use the following option, allowing zero-padding (i.e., the
    default operation) will make the output WARP dataset(s) be on a

```

(continues on next page)

(continued from previous page)

```

    larger grid (also see '-expad' below).
    argument: ``-nopad``
quiet: (a boolean)
    Cut out most of the fun fun fun progress messages :-(
    argument: ``-quiet``
    mutually_exclusive: verb
wmask: (a tuple of the form: (an existing file name, a float))
    -wmask ws fSimilar to '-wball', but here, you provide a dataset
    'ws' that indicates where to increase the weight.* The 'ws' dataset
    must be on the same 3D grid as the base dataset.* 'ws' is treated as
    a mask -- it only matters where it is nonzero -- otherwise, the
    values inside are not used.* After 'ws' comes the factor 'f' by
    which to increase the automatically computed weight. Where 'ws' is
    nonzero, the weighting will be multiplied by (1+f).* As with
    '-wball', the factor 'f' should be between 1 and 100.* You cannot
    use '-wball' and '-wmask' together!
    argument: ``-wpass %s %f``
emask: (an existing file name)
    Here, 'ee' is a dataset to specify a mask of voxels to EXCLUDE from
    the analysis -- all voxels in 'ee' that are NONZERO will not be used
    in the alignment.* The base image always automasked -- the emask
    is extra, to indicate voxels you definitely DON'T want included in the
    matching process, even if they are inside the brain.
    argument: ``-emask %s``
nowarp: (a boolean)
    Do not save the _WARP file.
    argument: ``-nowarp``
Qfinal: (a boolean)
    At the finest patch size (the final level), use Hermite quintic
    polynomials for the warp instead of cubic polynomials.* In a 3D
    'patch', there are 2x2x2x3=24 cubic polynomial basis function
    parameters over which to optimize (2 polynomials dependent on each of
    the x,y,z directions, and 3 different directions of displacement).*
    There are 3x3x3x3=81 quintic polynomial parameters per patch.* With
    -Qfinal, the final level will have more detail in the allowed warps,
    at the cost of yet more CPU time.* However, no patch below 7x7x7 in
    size will be done with quintic polynomials.* This option is also not
    usually needed, and is experimental.
    argument: ``-Qfinal``
resample: (a boolean)
    This option simply resamples the source dataset to match the base
    dataset grid. You can use this if the two datasets overlap well (as
    seen in the AFNI GUI), but are not on the same 3D grid.* If they
    don't overlap well, align them first.* The resampling here is done
    with the 'wsinc5' method, which has very little blurring artifact.*
    If the base and source datasets ARE on the same 3D grid, then the
    -resample option will be ignored.* You CAN use -resample with these
    3dQwarp options: -plusminus -inilev -iniwarp -duplo
    argument: ``-resample``
out_file: (a file name, nipy default value: Qwarp.nii.gz)
    Output file
    argument: ``-prefix %s``, position: 0
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
pblur: (a list of from 1 to 2 items which are a float)
    Use progressive blurring; that is, for larger patch sizes, the amount
    of blurring is larger. The general idea is to avoid trying to match

```

(continues on next page)

(continued from previous page)

```

finer details when the patch size and incremental warps are coarse.
When '-blur' is used as well, it sets a minimum amount of blurring
that will be used. [06 Aug 2014 -- '-pblur' may become the default
someday].* You can optionally give the fraction of the patch size
that is used for the progressive blur by providing a value between 0
and 0.25 after '-pblur'. If you provide TWO values, the first
fraction is used for progressively blurring the base image and the
second for the source image. The default parameters when just
'-pblur' is given is the same as giving the options as '-pblur 0.09
0.09'.* '-pblur' is useful when trying to match 2 volumes with
high amounts of detail; e.g., warping one subject's brain image
to match another's, or trying to warp to match a detailed template.*
Note that using negative values with '-blur' means that
the progressive blurring will be done with median filters, rather than
Gaussian linear blurring.-->*** The combination of the -allineate
and -pblur options will make the results of using 3dQwarp to align to
a template somewhat less sensitive to initial head position and
scaling.
argument: ``-pblur %s``
verb: (a boolean)
more detailed description of the process
argument: ``-verb``
mutually_exclusive: quiet
weight: (an existing file name)
Instead of computing the weight from the base dataset, directly input
the weight volume from dataset 'www'.* Useful if you know what over
parts of the base image you want to emphasize or de-emphasize the
matching functional.
argument: ``-weight %s``
allsave: (a boolean)
This option lets you save the output warps from each level of the
refinement process. Mostly used for experimenting.* Cannot be used
with -nopadWARP, -duplo, or -plusminus.* Will only save all the
outputs if the program terminates normally -- if it crashes, or
freezes, then all these warps are lost.
argument: ``-allsave``
mutually_exclusive: nopadWARP, duplo, plusminus
lpa: (a boolean)
Local Pearson maximization This option has not been extensively tested
argument: ``-lpa``
mutually_exclusive: nmi, mi, lpc, hel, pear
mi: (a boolean)
Mutual Information: a matching function for the adventurous This
option has NOT been extensively tested for usefulness and should be
considered experimental at this infundibulum.
argument: ``-mi``
mutually_exclusive: mi, hel, lpc, lpa, pear
nopadWARP: (a boolean)
If for some reason you require the warp volume to match the base
volume, then use this option to have the output WARP dataset(s)
truncated.
argument: ``-nopadWARP``
mutually_exclusive: allsave, expad
noweight: (a boolean)
If you want a binary weight (the old default), use this option. That
is, each voxel in the base volume auto-mask will be weighted the same
in the computation of the cost functional.

```

(continues on next page)

(continued from previous page)

```

argument: ``-noweight``
wball: (a list of from 5 to 5 items which are an integer (int or
       long))
       -wball x y z r fEnhance automatic weight from '-useweight' by a
       factor of 1+f*Gaussian(FWHM=r) centered in the base image at DICOM
       coordinates (x,y,z) and with radius 'r'. The goal of this option is
       to try and make the alignment better in a specific part of the
       brain.* Example: -wball 0 14 6 30 40 to emphasize the thalamic area
       (in MNI/Talairach space).* The 'r' parameter must be positive!* The
       'f' parameter must be between 1 and 100 (inclusive).* '-wball' does
       nothing if you input your own weight with the '-weight' option.*
       '-wball' does change the binary weight created by the '-noweight'
       option.* You can only use '-wball' once in a run of 3dQwarp.*** The
       effect of '-wball' is not dramatic. The example above makes the
       average brain image across a collection of subjects a little sharper
       in the thalamic area, which might have some small value. If you care
       enough about alignment to use '-wball', then you should examine
       the results from 3dQwarp for each subject, to see if the alignments
       are good enough for your purposes.
argument: ``-wball %s``

```

Outputs:

```

warped_source: (a file name)
               Warped source file. If plusminus is used, this is the
               undistorted source file.
warped_base: (a file name)
             Undistorted base file.
base_warp: (a file name)
           Displacement in mm for the base image. If plus minus is used, this is
           the field susceptibility correction warp (in 'mm') for base image.
           This is only output if plusminus or iwarp options are passed
weights: (a file name)
         Auto-computed weight volume.
source_warp: (a file name)
             Displacement in mm for the source image. If plusminus is used this is
             the field susceptibility correction warp (in 'mm') for source image.

```

References:

None None

51.3.24 ROIStats[Link to code](#)

Wraps the executable command 3dROIstats.

Display statistics over masked regions

For complete details, see the [3dROIstats Documentation](#)**Examples**

```

>>> from nipy.interfaces import afni
>>> roistats = afni.ROIStats()
>>> roistats.inputs.in_file = 'functional.nii'
>>> roistats.inputs.mask_file = 'skeleton_mask.nii.gz'

```

(continues on next page)

(continued from previous page)

```
>>> roistats.inputs.stat = ['mean', 'median', 'voxels']
>>> roistats.inputs.nomeanout = True
>>> roistats.cmdline
'3dROIstats -mask skeleton_mask.nii.gz -nomeanout -nzmean -nzmedian -nzvoxels_
↳functional.nii > functional_roistat.1D'
>>> res = roistats.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input dataset
        argument: ``%s``, position: -2

[Optional]
mask: (an existing file name)
      input mask
      argument: ``-mask %s``, position: 3
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
format1D: (a boolean)
          Output results in a 1D format that includes commented labels
          argument: ``-1Dformat``
          mutually_exclusive: format1DR
quiet: (a boolean)
       execute quietly
       argument: ``-quiet``
stat: (a list of items which are 'mean' or 'sum' or 'voxels' or
      'minmax' or 'sigma' or 'median' or 'mode' or 'summary' or
      'zerominmax' or 'zerosigma' or 'zeromedian' or 'zeromode')
      statistics to compute. Options include: * mean = Compute the mean
      using only non_zero voxels. Implies the opposite for the mean
      computed by default.
      * median = Compute the median of nonzero voxels
      * mode = Compute the mode of nonzero voxels. (integral valued sets
      only)
      * minmax = Compute the min/max of nonzero voxels
      * sum = Compute the sum using only nonzero voxels.
      * voxels = Compute the number of nonzero voxels
      * sigma = Compute the standard deviation of nonzero voxels
      Statistics that include zero-valued voxels:
      * zerominmax = Compute the min/max of all voxels.
      * zerosigma = Compute the standard deviation of all voxels.
      * zeromedian = Compute the median of all voxels.
      * zeromode = Compute the mode of all voxels.
      * summary = Only output a summary line with the grand mean across
      all briks in the input dataset. This option cannot be used with
      nomeanout.
      More that one option can be specified.
      argument: ``%s...``
out_file: (a file name)
          output file
          argument: ``> %s``, position: -1
roisel: (a file name)
        Only considers ROIs denoted by values found in the specified file.
        Note that the order of the ROIs as specified in the file is not
```

(continues on next page)

(continued from previous page)

```

    preserved. So an SEL1D of '2 8 20' produces the same output as '8
    20 2'
    argument: ``-roisel %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
mask_f2short: (a boolean)
    Tells the program to convert a float mask to short integers, by
    simple rounding.
    argument: ``-mask_f2short``
debug: (a boolean)
    print debug information
    argument: ``-debug``
nobriklab: (a boolean)
    Do not print the sub-brick label next to its index
    argument: ``-nobriklab``
num_roi: (an integer (int or long))
    Forces the assumption that the mask dataset's ROIs are denoted by 1
    to n inclusive. Normally, the program figures out the ROIs on its
    own. This option is useful if a) you are certain that the mask
    dataset has no values outside the range [0 n], b) there may be some
    ROIs missing between [1 n] in the mask data-set and c) you want
    those columns in the output any-way so the output lines up with the
    output from other invocations of 3dROIstats.
    argument: ``-numroi %s``
mask_file: (an existing file name)
    input mask
    argument: ``-mask %s``
nomeanout: (a boolean)
    Do not include the (zero-inclusive) mean among computed stats
    argument: ``-nomeanout``
zerofill: (a unicode string)
    For ROI labels not found, use the provided string instead of a '0'
    in the output file. Only active if `num_roi` is enabled.
    argument: ``-zerofill %s``
    requires: num_roi
format1DR: (a boolean)
    Output results in a 1D format that includes uncommented labels. May
    not work optimally with typical 1D functions, but is useful for R
    functions.
    argument: ``-1DRformat``
mutually_exclusive: format1D

```

Outputs:

```

out_file: (an existing file name)
    output tab-separated values file

```

51.3.25 Retroicor[Link to code](#)

Wraps the executable 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

simplicistic, 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  on](#).

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
         argument: ``%s``, position: -1

[Optional]
out_file: (a file name)
         output image file name
         argument: ``-prefix %s``, position: 1
respphase: (a file name)
         Filename for 1D resp phase output
         argument: ``-respphase %s``, position: -7
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
resp: (an existing file name)
      1D respiratory waveform data for correction
      argument: ``-resp %s``, position: -3
order: (an integer (int or long))
      The order of the correction (2 is typical)
      argument: ``-order %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
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
           AFNI output filetype
card: (an existing file name)
      1D cardiac data file for cardiac correction
      argument: ``-card %s``, position: -2
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
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)
```

(continues on next page)

(continued from previous page)

```

        argument: ``-threshold %d``, position: -4
cardphase: (a file name)
    Filename for 1D cardiac phase output
    argument: ``-cardphase %s``, position: -6

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.3.26 Seg[Link to code](#)

Wraps the executable 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]
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.
    argument: ``-mask %s``, position: -2
in_file: (an existing file name)
    ANAT is the volume to segment
    argument: ``-anat %s``, position: -1

[Optional]
blur_meth: ('BFT' or 'BIM')
    set the blurring method for bias field estimation
    argument: ``-blur_meth %s``
bmrfs: (a float)
    Weighting factor controlling spatial homogeneity of the
    classifications
    argument: ``-bmrfs %f``
classes: (a unicode string)
    CLASS_STRING is a semicolon delimited string of class labels
    argument: ``-classes %s``
main_N: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

        Number of iterations to perform.
        argument: ``-main_N %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
bias_fwhm: (a float)
        The amount of blurring used when estimating the field bias with the
        Wells method
        argument: ``-bias_fwhm %f``
mixfloor: (a float)
        Set the minimum value for any class's mixing fraction
        argument: ``-mixfloor %f``
prefix: (a unicode string)
        the prefix for the output folder containing all output volumes
        argument: ``-prefix %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
bias_classes: (a unicode string)
        A semicolon delimited string of classes that contribute to the
        estimation of the bias field
        argument: ``-bias_classes %s``
mixfrac: (a unicode string)
        MIXFRAC sets up the volume-wide (within mask) tissue fractions while
        initializing the segmentation (see IGNORE for exception)
        argument: ``-mixfrac %s``

```

Outputs:

```

out_file: (an existing file name)
        output file

```

51.3.27 SkullStrip[Link to code](#)

Wraps the executable 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

```

(continues on next page)

(continued from previous page)

```

        argument: ``-input %s``, position: 1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
out_file: (a file name)
           output image file name
           argument: ``-prefix %s``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
              set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables

```

Outputs:

```

out_file: (an existing file name)
          output file

```

References:

None None

51.3.28 TCorr1D[Link to code](#)

Wraps the executable 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_rclsl_Template_correlation.nii.gz u_rclsl_Template.nii
↪seed.1D'
>>> res = tcorr1D.run()

```

Inputs:

```

[Mandatory]
y_1d: (an existing file name)
      1D time series file input
      argument: `` %s``, position: -1
xset: (an existing file name)
      3d+time dataset input
      argument: `` %s``, position: -2

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

        Additional parameters to the command
        argument: ``%s``
out_file: (a file name)
        output filename prefix
        argument: ``-prefix %s``
pearson: (a boolean)
        Correlation is the normal Pearson correlation coefficient
        argument: ``-pearson``, position: 1
        mutually_exclusive: spearman, quadrant, ktaub
spearman: (a boolean)
        Correlation is the Spearman (rank) correlation coefficient
        argument: ``-spearman``, position: 1
        mutually_exclusive: pearson, quadrant, ktaub
num_threads: (an integer (int or long), nipy default value: 1)
        set number of threads
quadrant: (a boolean)
        Correlation is the quadrant correlation coefficient
        argument: ``-quadrant``, position: 1
        mutually_exclusive: pearson, spearman, ktaub
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
ktaub: (a boolean)
        Correlation is the Kendall's tau_b correlation coefficient
        argument: ``-ktaub``, position: 1
        mutually_exclusive: pearson, spearman, quadrant

```

Outputs:

```

out_file: (an existing file name)
        output file containing correlations

```

References:

None None

51.3.29 TCorrMap[Link to code](#)

Wraps the executable 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)
        argument: ``-input %s``

[Optional]
mean_file: (a file name)
        argument: ``-Mean %s``
correlation_maps: (a file name)
        argument: ``-CorrMap %s``
bandpass: (a tuple of the form: (a float, a float))
        argument: ``-bpass %f %f``
absolute_threshold: (a file name)
        argument: ``-Thresh %f %s``
        mutually_exclusive: absolute_threshold, var_absolute_threshold,
        var_absolute_threshold_normalize
average_expr: (a file name)
        argument: ``-Aexpr %s %s``
        mutually_exclusive: average_expr, average_expr_nonzero, sum_expr
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
pmean: (a file name)
        argument: ``-Pmean %s``
zmean: (a file name)
        argument: ``-Zmean %s``
automask: (a boolean)
        argument: ``-automask``
seeds_width: (a float)
        argument: ``-Mseed %f``
        mutually_exclusive: s, e, e, d, s
qmean: (a file name)
        argument: ``-Qmean %s``
sum_expr: (a file name)
        argument: ``-Sexpr %s %s``
        mutually_exclusive: average_expr, average_expr_nonzero, sum_expr
regress_out_timeseries: (a file name)
        argument: ``-ort %s``
var_absolute_threshold_normalize: (a file name)
        argument: ``-VarThreshN %f %f %f %s``
        mutually_exclusive: absolute_threshold, var_absolute_threshold,
        var_absolute_threshold_normalize
mask: (an existing file name)
        argument: ``-mask %s``
out_file: (a file name)
        output image file name
        argument: ``-prefix %s``
blur_fwhm: (a float)
        argument: ``-Gblur %f``
average_expr_nonzero: (a file name)
        argument: ``-Cexpr %s %s``
        mutually_exclusive: average_expr, average_expr_nonzero, sum_expr
num_threads: (an integer (int or long), nipy default value: 1)
        set number of threads
expr: (a unicode string)
correlation_maps_masked: (a file name)
        argument: ``-CorrMask %s``

```

(continues on next page)

(continued from previous page)

```

outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
thresholds: (a list of items which are an integer (int or long))
histogram_bin_numbers: (an integer (int or long))
histogram: (a file name)
            argument: ``-Hist %d %s``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
var_absolute_threshold: (a file name)
            argument: ``-VarThresh %f %f %s``
            mutually_exclusive: absolute_threshold, var_absolute_threshold,
            var_absolute_threshold_normalize
polort: (an integer (int or long))
        argument: ``-polort %d``
seeds: (an existing file name)
        argument: ``-seed %s``
        mutually_exclusive: s, e, e, d, s, _, w, i, d, t, h

```

Outputs:

```

mean_file: (a file name)
correlation_maps: (a file name)
var_absolute_threshold_normalize: (a file name)
absolute_threshold: (a file name)
var_absolute_threshold: (a file name)
zmean: (a file name)
average_expr: (a file name)
correlation_maps_masked: (a file name)
pmean: (a file name)
average_expr_nonzero: (a file name)
histogram: (a file name)
qmean: (a file name)
sum_expr: (a file name)

```

References:

None None

51.3.30 TCorrelate[Link to code](#)

Wraps the executable 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

```

(continues on next page)

(continued from previous page)

```
>>> 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
      argument: ``%s``, position: -2
yset: (an existing file name)
      input yset
      argument: ``%s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
out_file: (a file name)
           output image file name
           argument: ``-prefix %s``
pearson: (a boolean)
          Correlation is the normal Pearson correlation coefficient
          argument: ``-pearson``
polort: (an integer (int or long))
         Remove polynomial trend of order m
         argument: ``-polort %d``
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.3.31 TNorm[Link to code](#)

Wraps the executable command 3dTnorm.

Shifts voxel time series from input so that separate slices are aligned to the same temporal origin.

For complete details, see the [3dTnorm Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> tnorm = afni.TNorm()
>>> tnorm.inputs.in_file = 'functional.nii'
>>> tnorm.inputs.norm2 = True
>>> tnorm.inputs.out_file = 'rm.errts.unit errts+tlrc'
>>> tnorm.cmdline
'3dTnorm -norm2 -prefix rm.errts.unit errts+tlrc functional.nii'
>>> res = tshift.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dTNorm
         argument: ``%s``, position: -1

[Optional]
normR: (a boolean)
        normalize so sum of squares = number of time points * e.g., so RMS =
        1.
        argument: ``-normR``
out_file: (a file name)
           output image file name
           argument: ``-prefix %s``
num_threads: (an integer (int or long), nipy default value: 1)
              set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
normx: (a boolean)
        Scale so max absolute value = 1 (L_infinity norm)
        argument: ``-normx``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
             AFNI output filetype
L1fit: (a boolean)
        Detrend with L1 regression (L2 is the default)
        * This option is here just for the hell of it
        argument: ``-L1fit``
norm1: (a boolean)
        L1 normalize (sum of absolute values = 1)
        argument: ``-norm1``
polort: (an integer (int or long))
        Detrend with polynomials of order p before normalizing
        [DEFAULT = don't do this]
        * Use '-polort 0' to remove the mean, for example
        argument: ``-polort %s``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
norm2: (a boolean)
        L2 normalize (sum of squares = 1) [DEFAULT]
        argument: ``-norm2``
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.3.32 TProject

[Link to code](#)

Wraps the executable command 3dTproject.

This program projects (detrends) out various ‘nuisance’ time series from each voxel in the input dataset. Note that all the projections are done via linear regression, including the frequency-based options such as ‘-passband’. In this way, you can bandpass time-censored data, and at the same time, remove other time series of no interest (e.g., physiological estimates, motion parameters). Shifts voxel time series from input so that separate slices are aligned to the same temporal origin.

For complete details, see the [3dTproject Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> tproject = afni.TProject()
>>> tproject.inputs.in_file = 'functional.nii'
>>> tproject.inputs.bandpass = (0.00667, 99999)
>>> tproject.inputs.polort = 3
>>> tproject.inputs.automask = True
>>> tproject.inputs.out_file = 'projected.nii.gz'
>>> tproject.cmdline
'3dTproject -input functional.nii -automask -bandpass 0.00667 99999 -polort 3 -
↳prefix projected.nii.gz'
>>> res = tproject.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dTproject
         argument: ``-input %s``, position: 1

[Optional]
mask: (an existing file name)
      Only operate on voxels nonzero in the mset dataset.
      ++ Voxels outside the mask will be filled with zeros.
      ++ If no masking option is given, then all voxels
         will be processed.
      argument: ``-mask %s``
bandpass: (a tuple of the form: (a float, a float))
          Remove all frequencies EXCEPT those in the range
          argument: ``-bandpass %g %g``
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``, position: -1
concat: (an existing file name)
        The catenation file, as in 3dDeconvolve, containing the
        TR indexes of the start points for each contiguous run
        within the input dataset (the first entry should be 0).
```

(continues on next page)

(continued from previous page)

```

++ Also as in 3dDeconvolve, if the input dataset is
automatically catenated from a collection of datasets,
then the run start indexes are determined directly,
and '-concat' is not needed (and will be ignored).
++ Each run must have at least 9 time points AFTER
censoring, or the program will not work!
++ The only use made of this input is in setting up
the bandpass/stopband regressors.
++ '-ort' and '-dsort' regressors run through all time
points, as read in. If you want separate projections
in each run, then you must either break these ort files
into appropriate components, OR you must run 3dTproject
for each run separately, using the appropriate pieces
from the ort files via the '{...}' selector for the
1D files and the '[...]' selector for the datasets.
argument: ``-concat %s``
num_threads: (an integer (int or long), nipy default value: 1)
set number of threads
blur: (a float)
Blur (inside the mask only) with a filter that has
width (FWHM) of fff millimeters.
++ Spatial blurring (if done) is after the time
series filtering.
argument: ``-blur %g``
cenmode: ('KILL' or 'ZERO' or 'NTRP')
specifies how censored time points are treated in
the output dataset:
+ mode = ZERO ==> put zero values in their place
==> output dataset is same length as input
+ mode = KILL ==> remove those time points
==> output dataset is shorter than input
+ mode = NTRP ==> censored values are replaced by interpolated
neighboring (in time) non-censored values,
BEFORE any projections, and then the
analysis proceeds without actual removal
of any time points -- this feature is to
keep the Spanish Inquisition happy.
* The default mode is KILL !!!
argument: ``-cenmode %s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipy default value: {})
Environment variables
TR: (a float)
Use time step dd for the frequency calculations,
rather than the value stored in the dataset header.
argument: ``-TR %g``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
AFNI output filetype
censortr: (a list of items which are a unicode string)
list of strings that specify time indexes
to be removed from the analysis. Each string is
of one of the following forms:
37 => remove global time index #37
2:37 => remove time index #37 in run #2
37..47 => remove global time indexes #37-47
37-47 => same as above

```

(continues on next page)

(continued from previous page)

```

2:37..47 => remove time indexes #37-47 in run #2
*:0-2 => remove time indexes #0-2 in all runs
+Time indexes within each run start at 0.
+Run indexes start at 1 (just be to confusing).
+N.B.: 2:37,47 means index #37 in run #2 and
global time index 47; it does NOT mean
index #37 in run #2 AND index #47 in run #2.
argument: ``-CENSORTR %s``
censor: (an existing file name)
filename of censor .1D time series
* This is a file of 1s and 0s, indicating which
time points are to be included (1) and which are
to be excluded (0).
argument: ``-censor %s``
ort: (an existing file name)
Remove each column in file
++ Each column will have its mean removed.
argument: ``-ort %s``
dsort: (a list of items which are an existing file name)
Remove the 3D+time time series in dataset fset.
++ That is, 'fset' contains a different nuisance time
series for each voxel (e.g., from Anaticor).
++ Multiple -dsort options are allowed.
argument: ``-dsort %s...``
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
noblock: (a boolean)
Also as in 3dDeconvolve, if you want the program to treat
an auto-catenated dataset as one long run, use this option.
++ However, '-noblock' will not affect catenation if you use
the '-concat' option.
argument: ``-noblock``
automask: (a boolean)
Generate a mask automatically
argument: ``-automask``
mutually_exclusive: mask
polort: (an integer (int or long))
Remove polynomials up to and including degree pp.
++ Default value is 2.
++ It makes no sense to use a value of pp greater than
2, if you are bandpassing out the lower frequencies!
++ For catenated datasets, each run gets a separate set
set of pp+1 Legendre polynomial regressors.
++ Use of -polort -1 is not advised (if data mean != 0),
even if -ort contains constant terms, as all means are
removed.
argument: ``-polort %d``
stopband: (a tuple of the form: (a float, a float))
Remove all frequencies in the range
argument: ``-stopband %g %g``
norm: (a boolean)
Normalize each output time series to have sum of
squares = 1. This is the LAST operation.
argument: ``-norm``

```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.3.33 TShift

[Link to code](#)

Wraps the executable 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

Slice timing details may be specified explicitly via the `slice_timing` input:

```
>>> from nipy.interfaces import afni
>>> TR = 2.5
>>> tshift = afni.TShift()
>>> tshift.inputs.in_file = 'functional.nii'
>>> tshift.inputs.tzero = 0.0
>>> tshift.inputs.tr = '%.1fs' % TR
>>> tshift.inputs.slice_timing = list(np.arange(40) / TR)
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern @slice_timing.1D -TR 2.5s -tzero 0.
↳0 functional.nii'
```

When the `slice_timing` input is used, the `timing_file` output is populated, in this case with the generated file.

```
>>> tshift._list_outputs()['timing_file']
'.../slice_timing.1D'
```

```
>>> np.loadtxt(tshift._list_outputs()['timing_file']).tolist()[:5]
[0.0, 0.4, 0.8, 1.2, 1.6]
```

If `slice_encoding_direction` is set to `'k-'`, the slice timing is reversed:

```
>>> tshift.inputs.slice_encoding_direction = 'k-'
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern @slice_timing.1D -TR 2.5s -tzero 0.
↳0 functional.nii'
>>> np.loadtxt(tshift._list_outputs()['timing_file']).tolist()[:5]
[15.6, 15.2, 14.8, 14.4, 14.0]
```

This method creates a `slice_timing.1D` file to be passed to 3dTshift. A pre-existing slice-timing file may be used in the same way:

```
>>> tshift = afni.TShift()
>>> tshift.inputs.in_file = 'functional.nii'
>>> tshift.inputs.tzero = 0.0
>>> tshift.inputs.tr = '%.1fs' % TR
>>> tshift.inputs.slice_timing = 'slice_timing.1D'
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern @slice_timing.1D -TR 2.5s -tzero 0.
↳0 functional.nii'
```

When a pre-existing file is provided, `timing_file` is simply passed through.

```
>>> tshift._list_outputs()['timing_file']
'.../slice_timing.1D'
```

Alternatively, pre-specified slice timing patterns may be specified with the `tpattern` input. For example, to specify an alternating, ascending slice timing pattern:

```
>>> tshift = afni.TShift()
>>> tshift.inputs.in_file = 'functional.nii'
>>> tshift.inputs.tzero = 0.0
>>> tshift.inputs.tr = '%.1fs' % TR
>>> tshift.inputs.tpattern = 'alt+z'
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern alt+z -TR 2.5s -tzero 0.0_
↪functional.nii'
```

For backwards compatibility, `tpattern` may also take filenames prefixed with `@`. However, in this case, filenames are not validated, so this usage will be deprecated in future versions of Nipype.

```
>>> tshift = afni.TShift()
>>> tshift.inputs.in_file = 'functional.nii'
>>> tshift.inputs.tzero = 0.0
>>> tshift.inputs.tr = '%.1fs' % TR
>>> tshift.inputs.tpattern = '@slice_timing.1D'
>>> tshift.cmdline
'3dTshift -prefix functional_tshift -tpattern @slice_timing.1D -TR 2.5s -tzero 0.
↪0 functional.nii'
```

In these cases, `timing_file` is undefined.

```
>>> tshift._list_outputs()['timing_file']
<undefined>
```

In any configuration, the interface may be run as usual:

```
>>> res = tshift.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dTshift
         argument: ``%s``, position: -1

[Optional]
rlt: (a boolean)
     Before shifting, remove the mean and linear trend
     argument: ``-rlt``

slice_encoding_direction: ('k' or 'k-', nipype default value: k)
     Direction in which slice_timing is specified (default: k). If
     negative, slice_timing is defined in reverse order, that is, the
     first entry corresponds to the slice with the largest index, and the
     final entry corresponds to slice index zero. Only in effect when
     slice_timing is passed as list, not when it is passed as file.

slice_timing: (an existing file name or a list of items which are a
              float)
              time offsets from the volume acquisition onset for each slice
              argument: ``-tpattern %s``
              mutually_exclusive: tpattern

out_file: (a file name)
```

(continues on next page)

(continued from previous page)

```

        output image file name
        argument: ``-prefix %s``
ignore: (an integer (int or long))
        ignore the first set of points specified
        argument: ``-ignore %s``
rltplus: (a boolean)
        Before shifting, remove the mean and linear trend and later put back
        the mean
        argument: ``-rlt+``
num_threads: (an integer (int or long), nipy default value: 1)
        set number of threads
tslice: (an integer (int or long))
        align each slice to time offset of given slice
        argument: ``-slice %s``
        mutually_exclusive: tzero
tr: (a unicode string)
        manually set the TR. You can attach suffix "s" for seconds or "ms"
        for milliseconds.
        argument: ``-TR %s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
interp: ('Fourier' or 'linear' or 'cubic' or 'quintic' or 'heptic')
        different interpolation methods (see 3dTshift for details) default =
        Fourier
        argument: ``-%s``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
tpattern: ('alt+z' or 'altplus' or 'alt+z2' or 'alt-z' or 'altminus'
        or 'alt-z2' or 'seq+z' or 'seqplus' or 'seq-z' or 'seqminus' or a
        unicode string)
        use specified slice time pattern rather than one in header
        argument: ``-tpattern %s``
        mutually_exclusive: slice_timing
tzero: (a float)
        align each slice to given time offset
        argument: ``-tzero %s``
        mutually_exclusive: tslice
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``

```

Outputs:

```

timing_file: (a file name)
        AFNI formatted timing file, if ``slice_timing`` is a list
out_file: (an existing file name)
        output file

```

References:

None None

51.3.34 Volreg[Link to code](#)

Wraps the executable 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 -maxdisplD functional_md.1D functional.
↳nii'
>>> res = volreg.run()
```

```
>>> from nipy.interfaces import afni
>>> volreg = afni.Volreg()
>>> volreg.inputs.in_file = 'functional.nii'
>>> volreg.inputs.interp = 'cubic'
>>> volreg.inputs.verbose = True
>>> volreg.inputs.zpad = 1
>>> volreg.inputs.basefile = 'functional.nii'
>>> volreg.inputs.out_file = 'rm.epi.volreg.r1'
>>> volreg.inputs.oned_file = 'dfile.r1.1D'
>>> volreg.inputs.oned_matrix_save = 'mat.r1.tshift+orig.1D'
>>> volreg.cmdline
'3dvolreg -cubic -1Dfile dfile.r1.1D -1Dmatrix_save mat.r1.tshift+orig.1D -prefix_
↳rm.epi.volreg.r1 -verbose -base functional.nii -zpad 1 -maxdisplD functional_md.
↳1D functional.nii'
>>> res = volreg.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dvolreg
         argument: ``%s``, position: -1

[Optional]
out_file: (a file name)
         output image file name
         argument: ``-prefix %s``
basefile: (an existing file name)
         base file for registration
         argument: ``-base %s``, position: -6
timeshift: (a boolean)
         time shift to mean slice time offset
         argument: ``-tshift 0``
num_threads: (an integer (int or long), nipy default value: 1)
         set number of threads
oned_file: (a file name)
         1D movement parameters output file
         argument: ``-1Dfile %s``
environ: (a dictionary with keys which are a bytes or None or a value
```

(continues on next page)

(continued from previous page)

```

        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
    Environment variables
    interp: ('Fourier' or 'cubic' or 'heptic' or 'quintic' or 'linear')
        spatial interpolation methods [default = heptic]
        argument: ``-%s``
    outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
    mdl_d_file: (a file name)
        max displacement output file
        argument: ``-maxdisplD %s``, position: -4
    in_weight_volume: (a tuple of the form: (an existing file name, an
        integer (int or long)) or an existing file name)
        weights for each voxel specified by a file with an optional volume
        number (defaults to 0)
        argument: ``-weight '%s[%d]''``
    copyorigin: (a boolean)
        copy base file origin coords to output
        argument: ``-twodup``
    args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
    zpad: (an integer (int or long))
        Zeropad around the edges by 'n' voxels during rotations
        argument: ``-zpad %d``, position: -5
    verbose: (a boolean)
        more detailed description of the process
        argument: ``-verbose``
    oned_matrix_save: (a file name)
        Save the matrix transformation
        argument: ``-lDmatrix_save %s``

```

Outputs:

```

    oned_matrix_save: (an existing file name)
        matrix transformation from base to input
    mdl_d_file: (an existing file name)
        max displacement info file
    oned_file: (an existing file name)
        movement parameters info file
    out_file: (an existing file name)
        registered file

```

References:

None None

51.3.35 Warp[Link to code](#)

Wraps the executable 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
         argument: ``%s``, position: -1

[Optional]
deoblique: (a boolean)
           transform dataset from oblique to cardinal
           argument: ``-deoblique``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
save_warp: (a boolean)
           save warp as .mat file
           requires: verbose
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
tta2mni: (a boolean)
          transform dataset from Talairach to MNI152
          argument: ``-tta2mni``
environ: (a dictionary with keys which are a bytes or None or a 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
         argument: ``-gridset %s``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
matparent: (an existing file name)
            apply transformation from 3dWarpDrive
            argument: ``-matparent %s``
newgrid: (a float)
          specify grid of this size (mm)
```

(continues on next page)

(continued from previous page)

```

        argument: ``-newgrid %f``
verbose: (a boolean)
        Print out some information along the way.
        argument: ``-verb``
zpad: (an integer (int or long))
        pad input dataset with N planes of zero on all sides.
        argument: ``-zpad %d``
interp: ('linear' or 'cubic' or 'NN' or 'quintic')
        spatial interpolation methods [default = linear]
        argument: ``-%s``
mni2tta: (a boolean)
        transform dataset from MNI152 to Talaraich
        argument: ``-mni2tta``
oblique_parent: (an existing file name)
        Read in the oblique transformation matrix from an oblique dataset
        and make cardinal dataset oblique to match
        argument: ``-oblique_parent %s``

```

Outputs:

```

out_file: (an existing file name)
        Warped file.
warp_file: (a file name)
        warp transform .mat file

```

References:

None None

51.4 interfaces.afni.svm

51.4.1 SVMTest

[Link to code](#)

Wraps the executable 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 brik dataset to be used for testing.
        argument: ``-testvol %s``
model: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

        modname is the basename for the briik containing the SVM model
        argument: ``-model %s``

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
out_file: (a file name)
    filename for .1D prediction file(s).
    argument: ``-predictions %s``
nopredcensord: (a boolean)
    Flag to prevent writing predicted values for censored time-points
    argument: ``-nopredcensord``
nodetrend: (a boolean)
    Flag to specify that pname files should not be linearly detrended
    argument: ``-nodetrend``
multiclass: (a boolean)
    Specifies multiclass algorithm for classification
    argument: ``-multiclass %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
classout: (a boolean)
    Flag to specify that pname files should be integer-valued,
    corresponding to class category decisions.
    argument: ``-classout``
options: (a unicode string)
    additional options for SVM-light
    argument: ``%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
testlabels: (an existing file name)
    *true* class category .1D labels for the test dataset. It is used to
    calculate the prediction accuracy performance
    argument: ``-testlabels %s``

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.4.2 SVMTrain[Link to code](#)

Wraps the executable 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.
    argument: ``-trainvol %s``
ttype: (a unicode string)
    tname: classification or regression
    argument: ``-type %s``

[Optional]
trainlabels: (an existing file name)
    .1D labels corresponding to the stimulus paradigm for the training
    data.
    argument: ``-trainlabels %s``
mask: (an existing file name)
    byte-format brik file used to mask voxels in the analysis
    argument: ``-mask %s``, position: -1
max_iterations: (an integer (int or long))
    Specify the maximum number of iterations for the optimization.
    argument: ``-max_iterations %d``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
w_out: (a boolean)
    output sum of weighted linear support vectors
    argument: ``-wout``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
censor: (an existing file name)
    .1D censor file that allows the user to ignore certain samples in
    the training data.
    argument: ``-censor %s``
options: (a unicode string)
    additional options for SVM-light
    argument: ``%s``
model: (a file name)
    basename for the brik containing the SVM model
    argument: ``-model %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
kernel: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

        string specifying type of kernel function: linear, polynomial, rbf,
        sigmoid
        argument: ``-kernel %s``
alphas: (a file name)
        output alphas file name
        argument: ``-alpha %s``
nomodelmask: (a boolean)
        Flag to enable the omission of a mask file
        argument: ``-nomodelmask``
out_file: (a file name)
        output sum of weighted linear support vectors file name
        argument: ``-bucket %s``

```

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

51.5 interfaces.afni.utils

51.5.1 ABoverlap

[Link to code](#)

Wraps the executable command 3dABoverlap.

Output (to screen) is a count of various things about how the automasks of datasets A and B overlap or don't overlap.

For complete details, see the [3dABoverlap Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> aboverlap = afni.ABoverlap()
>>> aboverlap.inputs.in_file_a = 'functional.nii'
>>> aboverlap.inputs.in_file_b = 'structural.nii'
>>> aboverlap.inputs.out_file = 'out.mask_ae_overlap.txt'
>>> aboverlap.cmdline
'3dABoverlap functional.nii structural.nii |& tee out.mask_ae_overlap.txt'
>>> res = aboverlap.run()

```

Inputs:

```

[Mandatory]
in_file_b: (an existing file name)
        input file B
        argument: ``%s``, position: -2
in_file_a: (an existing file name)
        input file A

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``, position: -3

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
out_file: (a file name)
    collect output to a file
    argument: ``|& tee %s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
quiet: (a boolean)
    be as quiet as possible (without being entirely mute)
    argument: ``-quiet``
verb: (a boolean)
    print out some progress reports (to stderr)
    argument: ``-verb``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
no_automask: (a boolean)
    consider input datasets as masks
    argument: ``-no_automask``

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.5.2 AFNItoNIFTI[Link to code](#)

Wraps the executable command 3dAFNItoNIFTI.

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 [3dAFNItoNIFTI 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
        argument: ``%s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
denote: (a boolean)
        When writing the AFNI extension field, remove text notes that might
        contain subject identifying information.
        argument: ``-denote``
oldid: (a boolean)
        Give the new dataset the input datasets AFNI ID code.
        argument: ``-oldid``
        mutually_exclusive: newid
out_file: (a file name)
        output image file name
        argument: ``-prefix %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
newid: (a boolean)
        Give the new dataset a new AFNI ID code, to distinguish it from the
        input dataset.
        argument: ``-newid``
        mutually_exclusive: oldid
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.
        argument: ``-pure``
num_threads: (an integer (int or long), nipy default value: 1)
            set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:

None None

51.5.3 Autobox[Link to code](#)

Wraps the executable 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 nipy.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
        argument: ``-input %s``

[Optional]
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
         argument: ``-npad %d``
out_file: (a file name)
         argument: ``-prefix %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
            set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
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.
              argument: ``-noclust``
```

Outputs:

```
y_min: (an integer (int or long))
z_max: (an integer (int or long))
x_min: (an integer (int or long))
out_file: (a file name)
          output file
y_max: (an integer (int or long))
x_max: (an integer (int or long))
z_min: (an integer (int or long))
```

References:

None None

51.5.4 Axialize

[Link to code](#)

Wraps the executable command 3daxialize.

Read in a dataset and write it out as a new dataset with the data brick oriented as axial slices.

For complete details, see the [3dcopy Documenta­tion](#).

Examples

```
>>> from nipy.interfaces import afni
>>> axial3d = afni.Axialize()
>>> axial3d.inputs.in_file = 'functional.nii'
>>> axial3d.inputs.out_file = 'axialized.nii'
>>> axial3d.cmdline
'3daxialize -prefix axialized.nii functional.nii'
>>> res = axial3d.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3daxialize
         argument: ``%s``, position: -2

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
orientation: (a unicode string)
             new orientation code
             argument: ``-orient %s``
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
axial: (a boolean)
       Do axial slice order [-orient RAI]This is the default AFNI axial
       order, andis the one currently required by thevolume rendering
       plugin; this is alsothe default orientation output by thisprogram
       (hence the program's name).
       argument: ``-axial``
       mutually_exclusive: coronal, sagittal
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
verb: (a boolean)
      Print out a progerss report
      argument: ``-verb``
sagittal: (a boolean)
          Do sagittal slice order [-orient ASL]
          argument: ``-sagittal``
          mutually_exclusive: coronal, axial
coronal: (a boolean)
          Do coronal slice order [-orient RSA]
          argument: ``-coronal``
          mutually_exclusive: sagittal, axial
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.5.5 BrickStat[Link to code](#)

Wraps the executable 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
          argument: ``%s``, position: -1

[Optional]
mask: (an existing file name)
      -mask dset = use dset as mask to include/exclude voxels
      argument: ``-mask %s``, position: 2
mean: (a boolean)
      print the mean value in the dataset
      argument: ``-mean``
slow: (a boolean)
      read the whole dataset to find the min and max values
      argument: ``-slow``
min: (a boolean)
      print the minimum value in dataset
      argument: ``-min``, position: 1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
sum: (a boolean)
      print the sum of values in the dataset
      argument: ``-sum``
var: (a boolean)
      print the variance in the dataset
      argument: ``-var``
percentile: (a tuple of the form: (a float, a float, a float))
            p0 ps pl write the percentile values starting at p0% and ending at
```

(continues on next page)

(continued from previous page)

```

    p1% at a step of ps%. only one sub-brick is accepted.
    argument: ``-percentile %.3f %.3f %.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
max: (a boolean)
    print the maximum value in the dataset
    argument: ``-max``

```

Outputs:

```

min_val: (a float)
    output

```

51.5.6 Bucket

[Link to code](#)

Wraps the executable command 3dbucket.

Concatenate sub-bricks from input datasets into one big 'bucket' dataset.

For complete details, see the [3dbucket Documentation](#).

Examples

```

>>> from nipy.interfaces import afni
>>> bucket = afni.Bucket()
>>> bucket.inputs.in_file = [('functional.nii', "{2..$}"), ('functional.nii', "{1}
↳")]
>>> bucket.inputs.out_file = 'vr_base'
>>> bucket.cmdline
"3dbucket -prefix vr_base functional.nii'{2..$}' functional.nii'{1}'"
>>> res = bucket.run()

```

Inputs:

```

[Mandatory]
in_file: (a list of items which are a tuple of the form: (an existing
    file name, a unicode string))
    List of tuples of input datasets and subbrick selection stringsas
    described in more detail in the following afni help stringInput
    dataset specified using one of these forms: 'prefix+view',
    'prefix+view.HEAD', or 'prefix+view.BRIK'.You can also add a sub-
    brick selection list after the end of thedataset name. This allows
    only a subset of the sub-bricks to beincluded into the output (by
    default, all of the input datasetis copied into the output). A sub-
    brick selection list looks likeone of the following forms:
    fred+orig[5] ==> use only sub-brick #5 fred+orig[5,9,17] ==> use #5,
    #9, and #17 fred+orig[5..8] or [5-8] ==> use #5, #6, #7, and #8
    fred+orig[5..13(2)] or [5-13(2)] ==> use #5, #7, #9, #11, and
    #13Sub-brick indexes start at 0. You can use the character '$'to
    indicate the last sub-brick in a dataset; for example, youcan select
    every third sub-brick by using the selection list
    fred+orig[0..$(3)]N.B.: The sub-bricks are output in the order
    specified, which may not be the order in the original datasets. For
    example, using fred+orig[0..$(2),1..$(2)] will cause the sub-bricks
    in fred+orig to be output into the new dataset in an interleaved

```

(continues on next page)

(continued from previous page)

fashion. Using `fred+orig[$..0]` will reverse the order of the sub-bricks in the output. N.B.: Bucket datasets have multiple sub-bricks, but do NOT have a time dimension. You can input sub-bricks from a 3D+time dataset into a bucket dataset. You can use the '3dinfo' program to see how many sub-bricks a 3D+time or a bucket dataset contains. N.B.: In non-bucket functional datasets (like the 'fico' datasets output by FIM, or the 'fitt' datasets output by 3dttest), sub-brick [0] is the 'intensity' and sub-brick [1] is the statistical parameter used as a threshold. Thus, to create a bucket dataset using the intensity from dataset A and the threshold from dataset B, and calling the output dataset C, you would type `3dbucket -prefix C -fbuc 'A+orig[0]' -fbuc 'B+orig[1]'` WARNING: using this program, it is possible to create a dataset that has different basic datum types for different sub-bricks (e.g., shorts for brick 0, floats for brick 1). Do NOT do this! Very few AFNI programs will work correctly with such datasets!

argument: ``%s``, position: -1

[Optional]
 outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
 AFNI output filetype
 out_file: (a file name)
 argument: ``-prefix %s``
 args: (a unicode string)
 Additional parameters to the command
 argument: ``%s``
 num_threads: (an integer (int or long), nipy default value: 1)
 set number of threads
 environ: (a dictionary with keys which are a bytes or None or a value
 of class 'str' and with values which are a bytes or None or a
 value of class 'str', nipy default value: {})
 Environment variables

Outputs:

out_file: (an existing file name)
 output file

References:

None None

51.5.7 Calc

[Link to code](#)

Wraps the executable 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'
```

(continues on next page)

(continued from previous page)

```
>>> 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()
```

```
>>> from nipy.interfaces import afni
>>> calc = afni.Calc()
>>> calc.inputs.in_file_a = 'functional.nii'
>>> calc.inputs.expr = '1'
>>> calc.inputs.out_file = 'rm.epi.all1'
>>> calc.inputs.overwrite = True
>>> calc.cmdline
'3dcalc -a functional.nii -expr "1" -prefix rm.epi.all1 -overwrite'
>>> res = calc.run()
```

Inputs:

```
[Mandatory]
expr: (a unicode string)
    expr
    argument: ``-expr "%s"`, position: 3
in_file_a: (an existing file name)
    input file to 3dcalc
    argument: ``-a %s`, position: 0

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
in_file_c: (an existing file name)
    operand file to 3dcalc
    argument: ``-c %s`, position: 2
out_file: (a file name)
    output image file name
    argument: ``-prefix %s`
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
single_idx: (an integer (int or long))
    volume index for in_file_a
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
stop_idx: (an integer (int or long))
    stop index for in_file_a
    requires: start_idx
other: (a file name)
    other options
in_file_b: (an existing file name)
    operand file to 3dcalc
    argument: ``-b %s`, position: 1
overwrite: (a boolean)
    overwrite output
    argument: ``-overwrite`
args: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

        Additional parameters to the command
        argument: ``%s``
start_idx: (an integer (int or long))
        start index for in_file_a
        requires: stop_idx

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:

None None

51.5.8 Cat[Link to code](#)Wraps the executable command `ldcat`.

`ldcat` takes as input one or more 1D files, and writes out a 1D file containing the side-by-side concatenation of all or a subset of the columns from the input files.

For complete details, see the [ldcat Documentation](#).

Examples

```

>>> from nipy.interfaces import afni
>>> catld = afni.Cat()
>>> catld.inputs.sel = "[0,2]"
>>> catld.inputs.in_files = ['f1.1D', 'f2.1D']
>>> catld.inputs.out_file = 'catout.1d'
>>> catld.cmdline
"ldcat -sel '[0,2]' f1.1D f2.1D > catout.1d"
>>> res = catld.run()

```

Inputs:

```

[Mandatory]
in_files: (a list of items which are an existing file name)
        argument: ``%s``, position: -2
out_file: (a file name, nipy default value: catout.1d)
        output (concatenated) file name
        argument: ``> %s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
out_format: ('int' or 'nice' or 'double' or 'fint' or 'cint')
        specify data type for output. Valid types are 'int', 'nice',
        'double', 'fint', and 'cint'.
        argument: ``-form %s``
        mutually_exclusive: out_int, out_nice, out_double, out_fint,
        out_cint
out_double: (a boolean)
        specify double data type for output
        argument: ``-d``
        mutually_exclusive: out_format, out_nice, out_int, out_fint,

```

(continues on next page)

(continued from previous page)

```

        out_cint
keepfree: (a boolean)
    Keep only columns that are marked as 'free' in the 3dAllineate
    header from '-1Dparam_save'. If there is no such header, all columns
    are kept.
    argument: ``-nonfixed``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
out_nice: (a boolean)
    specify nice data type for output
    argument: ``-n``
    mutually_exclusive: out_format, out_int, out_double, out_fint,
        out_cint
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
stack: (a boolean)
    Stack the columns of the resultant matrix in the output.
    argument: ``-stack``
out_fint: (a boolean)
    specify int, rounded down, data type for output
    argument: ``-f``
    mutually_exclusive: out_format, out_nice, out_double, out_int,
        out_cint
omitconst: (a boolean)
    Omit columns that are identically constant from output.
    argument: ``-nonconst``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
out_int: (a boolean)
    specify int data type for output
    argument: ``-i``
    mutually_exclusive: out_format, out_nice, out_double, out_fint,
        out_cint
sel: (a unicode string)
    Apply the same column/row selection string to all filenames on the
    command line.
    argument: ``-sel %s``
out_cint: (a boolean)
    specify int, rounded up, data type for output
    mutually_exclusive: out_format, out_nice, out_double, out_fint,
        out_int

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.5.9 CatMatvec[Link to code](#)

Wraps the executable command `cat_matvec`.
Catenates 3D rotation+shift matrix+vector transformations.
For complete details, see the [cat_matvec Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> cmv = afni.CatMatvec()
>>> cmv.inputs.in_file = [('structural.BRIK::WARP_DATA', 'I')]
>>> cmv.inputs.out_file = 'warp.anat.Xat.1D'
>>> cmv.cmdline
'cat_matvec structural.BRIK::WARP_DATA -I > warp.anat.Xat.1D'
>>> res = cmv.run()
```

Inputs:

```
[Mandatory]
in_file: (a list of items which are a tuple of the form: (a unicode
          string, a unicode string))
          list of tuples of mfiles and associated opkeys
          argument: ``%s``, position: -2
out_file: (a file name)
          File to write concattenated matvecs to
          argument: `` > %s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
fourxfour: (a boolean)
            Output matrix in augmented form (last row is 0 0 0 1)This option
            does not work with -MATRIX or -ONELINE
            argument: ``-4x4``
            mutually_exclusive: matrix, oneline
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
matrix: (a boolean)
         indicates that the resulting matrix willbe written to outfile in the
         'MATRIX(...)' format (FORM 3).This feature could be used, with
         clever scripting, to inputa matrix directly on the command line to
         program 3dWarp.
         argument: ``-MATRIX``
         mutually_exclusive: oneline, fourxfour
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
oneline: (a boolean)
         indicates that the resulting matrixwill simply be written as 12
         numbers on one line.
         argument: ``-ONELINE``
         mutually_exclusive: matrix, fourxfour
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.5.10 CenterMass

[Link to code](#)

Wraps the executable command 3dCM.

Computes center of mass using 3dCM command

Note: By default, the output is (x,y,z) values in DICOM coordinates. But as of Dec, 2016, there are now command line switches for other options.

For complete details, see the [3dCM Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> cm = afni.CenterMass()
>>> cm.inputs.in_file = 'structural.nii'
>>> cm.inputs.cm_file = 'cm.txt'
>>> cm.inputs.roi_vals = [2, 10]
>>> cm.cmdline
'3dCM -roi_vals 2 10 structural.nii > cm.txt'
>>> res = 3dcm.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dCM
         argument: ``%s``, position: -2

[Optional]
set_cm: (a tuple of the form: (a float, a float, a float))
        After computing the center of mass, set the origin fields in the
        header so that the center of mass will be at (x,y,z) in DICOM
        coords.
        argument: ``-set %f %f %f``
mask_file: (an existing file name)
           Only voxels with nonzero values in the provided mask will be
           averaged.
           argument: ``-mask %s``
all_rois: (a boolean)
          Don't bother listing the values of ROIs you want: The program will
          find all of them and produce a full list
          argument: ``-all_rois``
roi_vals: (a list of items which are an integer (int or long))
          Compute center of mass for each blob with voxel value of v0, v1, v2,
          etc. This option is handy for getting ROI centers of mass.
          argument: ``-roi_vals %s``
args: (a unicode string)
      Additional parameters to the command
```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
cm_file: (a file name)
        File to write center of mass to
        argument: ``> %s``, position: -1
automask: (a boolean)
        Generate the mask automatically
        argument: ``-automask``
local_ijk: (a boolean)
        Output values as (i,j,k) in local orientation
        argument: ``-local_ijk``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

cm: (a list of items which are a tuple of the form: (a float, a
        float, a float))
        center of mass
out_file: (an existing file name)
        output file
cm_file: (a file name)
        file with the center of mass coordinates

```

51.5.11 ConvertDset[Link to code](#)

Wraps the executable command ConvertDset.

Converts a surface dataset from one format to another.

For complete details, see the [ConvertDset Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> convert_dset = afni.ConvertDset()
>>> convert_dset.inputs.in_file = 'lh.pial_converted.gii'
>>> convert_dset.inputs.out_type = 'niml_asc'
>>> convert_dset.inputs.out_file = 'lh.pial_converted.niml.dset'
>>> convert_dset.cmdline
'ConvertDset -o_niml_asc -input lh.pial_converted.gii -prefix lh.pial_converted.
↳niml.dset'
>>> res = convert_dset.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file to ConvertDset
        argument: ``-input %s``, position: -2
out_type: ('niml' or 'niml_asc' or 'niml_bi' or '1D' or '1Dp' or
        '1Dpt' or 'gii' or 'gii_asc' or 'gii_b64' or 'gii_b64gz')
        output type
        argument: ``-o_%s``, position: 0
out_file: (a file name)
        output file for ConvertDset

```

(continues on next page)

(continued from previous page)

```

        argument: ``-prefix %s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

out_file: (an existing file name)
    output file

```

51.5.12 Copy[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: -2

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
out_file: (a file name)
          output image file name
          argument: ``%s``, position: -1
verbose: (a boolean)
         print progress reports
         argument: ``-verb``
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.5.13 Dot[Link to code](#)

Wraps the executable command 3dDot.

Correlation coefficient between sub-brick pairs. All datasets in in_files list will be concatenated. You can use sub-brick selectors in the file specification. Note: This program is not efficient when more than two subbricks are input. For complete details, see the [3ddot Documenta**tion**](#).

```
>>> from nipy.interfaces import afni
>>> dot = afni.Dot()
>>> dot.inputs.in_files = ['functional.nii[0]', 'structural.nii']
>>> dot.inputs.dodice = True
>>> dot.inputs.out_file = 'out.mask_ae_dice.txt'
>>> dot.cmdline
'3dDot -dodice functional.nii[0] structural.nii  |& tee out.mask_ae_dice.txt'
>>> res = copy3d.run()
```

Inputs:

```
[Optional]
mask: (a file name)
      Use this dataset as a mask
      argument: ``-mask %s``
```

(continues on next page)

(continued from previous page)

```

dosums: (a boolean)
    Return the 6 numbers  $\bar{x}$   $\bar{y}$   $\langle (x-\bar{x})^2 \rangle$   $\langle (y-\bar{y})^2 \rangle$ 
     $\langle (x-\bar{x})(y-\bar{y}) \rangle$  and the correlation coefficient.
    argument: ``-dosums``

upper: (a boolean)
    Compute upper triangular matrix
    argument: ``-upper``

dodot: (a boolean)
    Return the dot product (unscaled).
    argument: ``-dodot``

docoef: (a boolean)
    Return the least square fit coefficients  $\{a, b\}$  so that  $dset2$  is
    approximately  $a + b \cdot dset1$ 
    argument: ``-docoef``

out_file: (a file name)
    collect output to a file
    argument: ``|& tee %s``, position: -1

full: (a boolean)
    Compute the whole matrix. A waste of time, but handy for parsing.
    argument: ``-full``

in_files: (a list of items which are a file name)
    list of input files, possibly with subbrick selectors
    argument: ``%s ...``, position: -2

mrange: (a tuple of the form: (a float, a float))
    Means to further restrict the voxels from 'mset' so that only those
    mask values within this range (inclusive) will be used.
    argument: ``-mrange %s %s``

dodice: (a boolean)
    Return the Dice coefficient (the Sorensen-Dice index).
    argument: ``-dodice``

show_labels: (a boolean)
    Print sub-brick labels to help identify what is being correlated.
    This option is useful when you have more than 2 sub-bricks at input.
    argument: ``-show_labels``

outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype

docor: (a boolean)
    Return the correlation coefficient (default).
    argument: ``-docor``

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

doeta2: (a boolean)
    Return eta-squared (Cohen, NeuroImage 2008).
    argument: ``-doeta2``

demean: (a boolean)
    Remove the mean from each volume prior to computing the correlation
    argument: ``-demean``

num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads

```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.5.14 Edge3

[Link to code](#)

Wraps the executable command 3dedge3.

Does 3D Edge detection using the library 3DEdge by Gregoire Malandain (gregoire.malandain@sophia.inria.fr). For complete details, see the [3dedge3 Documenta  on](#).

```
references_ = [{‘entry’: BibTeX(‘@article{Deriche1987,’
                               ‘author={R. Deriche}, ‘title={Optimal edge detection using recursive filtering}, ‘jour-
                               nal={International Journal of Computer Vision}, ‘volume={2}, ‘pages={167-187}, ‘
                               ‘year={1987}, ‘}’),
               ‘tags’: [‘method’], },
               ‘entry’: BibTeX(‘@article{MongaDericheMalandainCocquerez1991,’
                               ‘author={O. Monga, R. Deriche, G. Malandain, J.P. Cocquerez}, ‘title={Recursive filtering
                               and edge tracking: two primary tools for 3D edge detection}, ‘journal={Image and vision
                               computing}, ‘volume={9}, ‘pages={203-214}, ‘year={1991}, ‘}’),
               ‘tags’: [‘method’], },
               ~
```

Examples

```
>>> from nipy.interfaces import afni
>>> edge3 = afni.Edge3()
>>> edge3.inputs.in_file = ‘functional.nii’
>>> edge3.inputs.out_file = ‘edges.nii’
>>> edge3.inputs.datum = ‘byte’
>>> edge3.cmdline
‘3dedge3 -input functional.nii -datum byte -prefix edges.nii’
>>> res = edge3.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3dedge3
         argument: ‘‘-input %s‘‘, position: 0

[Optional]
datum: (‘byte’ or ‘short’ or ‘float’)
       specify data type for output. Valid types are ‘byte’, ‘short’ and
       ‘float’.
       argument: ‘‘-datum %s‘‘
out_file: (a file name)
          output image file name
          argument: ‘‘-prefix %s‘‘, position: -1
fscale: (a boolean)
        Force scaling of the output to the maximum integer range.
        argument: ‘‘-fscale‘‘
        mutually_exclusive: gscale, nscale, scale_floats
```

(continues on next page)

(continued from previous page)

```

num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
scale_floats: (a float)
    Multiply input by VAL, but only if the input datum is float. This is
    needed when the input dataset has a small range, like 0 to 2.0 for
    instance. With such a range, very few edges are detected due to what
    I suspect to be truncation problems. Multiplying such a dataset by
    10000 fixes the problem and the scaling is undone at the output.
    argument: ``-scale_floats %f``
    mutually_exclusive: fscale, gscale, nscale
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    Print out some information along the way.
    argument: ``-verbose``
nscale: (a boolean)
    Don't do any scaling on output to byte or short datasets.
    argument: ``-nscale``
    mutually_exclusive: fscale, gscale, scale_floats
gscale: (a boolean)
    Same as '-fscale', but also forces each output sub-brick to to get
    the same scaling factor.
    argument: ``-gscale``
    mutually_exclusive: fscale, nscale, scale_floats

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.5.15 Eval[Link to code](#)Wraps the executable command `ldeval`.

Evaluates an expression that may include columns of data from one or more text files.

For complete details, see the [ldeval 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'

```

(continues on next page)

(continued from previous page)

```
>>> eval.inputs.out1D = True
>>> eval.inputs.out_file = 'data_calc.1D'
>>> eval.cmdline
'ldeval -a seed.1D -b resp.1D -expr "a*b" -1D -prefix data_calc.1D'
>>> res = eval.run()
```

Inputs:

```
[Mandatory]
expr: (a unicode string)
    expr
    argument: ``-expr "%s"`, position: 3
in_file_a: (an existing file name)
    input file to ldeval
    argument: ``-a %s`, position: 0

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
in_file_c: (an existing file name)
    operand file to ldeval
    argument: ``-c %s`, position: 2
out_file: (a file name)
    output image file name
    argument: ``-prefix %s`
out1D: (a boolean)
    output in 1D
    argument: ``-1D`
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
single_idx: (an integer (int or long))
    volume index for in_file_a
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
stop_idx: (an integer (int or long))
    stop index for in_file_a
    requires: start_idx
other: (a file name)
    other options
in_file_b: (an existing file name)
    operand file to ldeval
    argument: ``-b %s`, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s`
start_idx: (an integer (int or long))
    start index for in_file_a
    requires: stop_idx
```

Outputs:

```
out_file: (an existing file name)
    output file
```

References:

None None

51.5.16 FWHMx[Link to code](#)

Wraps the executable 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
        argument: ``-input %s``

[Optional]
combine: (a boolean)
        combine the final measurements along each axis
        argument: ``-combine``
mask: (an existing file name)
        use only voxels that are nonzero in mask
        argument: ``-mask %s``
compat: (a boolean)
        be compatible with the older 3dFWHM
        argument: ``-compat``
out_file: (a file name)
        output file
        argument: ``> %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
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.
        argument: ``-unif``
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
```

(continues on next page)

(continued from previous page)

```

        structure and leave behind the noise.
        argument: ``-demed``
        mutually_exclusive: detrend
    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
        argument: ``-acf``
    out_detrend: (a file name)
        Save the detrended file into a dataset
        argument: ``-detprefix %s``
    args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
    automask: (a boolean, nipy default value: False)
        compute a mask from THIS dataset, a 1a 3dAutomask
        argument: ``-automask``
    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
        argument: ``-geom``
        mutually_exclusive: arith
    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
        argument: ``-arith``
        mutually_exclusive: geom
    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.
        argument: ``-detrend``
        mutually_exclusive: demed
    out_subbricks: (a file name)
        output file listing the subbricks FWHM
        argument: ``-out %s``

```

Outputs:

```

    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_detrend: (a file name)
        output file, detrended
    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
    out_subbricks: (an existing file name)
        output file (subbricks)
    out_acf: (an existing file name)
        output acf file
    out_file: (an existing file name)
        output file

```

References:

None

51.5.17 GCOR

[Link to code](#)

Wraps the executable command `@compute_gcor`.

Computes the average correlation between every voxel and every other voxel, over any given mask.

For complete details, see the [@compute_gcor Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> gcor = afni.GCOR()
>>> gcor.inputs.in_file = 'structural.nii'
>>> gcor.inputs.nfirst = 4
>>> gcor.cmdline
'@compute_gcor -nfirst 4 -input structural.nii'
>>> res = gcor.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input dataset to compute the GCOR over
         argument: ``-input %s``, position: -1

[Optional]
mask: (an existing file name)
      mask dataset, for restricting the computation
      argument: ``-mask %s``
no_demean: (a boolean)
            do not (need to) demean as first step
            argument: ``-no_demean``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
nfirst: (an integer (int or long))
        specify number of initial TRs to ignore
        argument: ``-nfirst %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
```

Outputs:

```
out: (a float)
      global correlation value
```

51.5.18 LocalBistat

[Link to code](#)

Wraps the executable command `3dLocalBistat`.

`3dLocalBistat` - computes statistics between 2 datasets, at each voxel, based on a local neighborhood of that voxel.

For complete details, see the [3dLocalBistat Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> bistat = afni.LocalBistat()
>>> bistat.inputs.in_file1 = 'functional.nii'
>>> bistat.inputs.in_file2 = 'structural.nii'
>>> bistat.inputs.neighborhood = ('SPHERE', 1.2)
>>> bistat.inputs.stat = 'pearson'
>>> bistat.inputs.outputtype = 'NIFTI'
>>> bistat.cmdline
"3dLocalBistat -prefix functional_bistat.nii -nbhd 'SPHERE(1.2)' -stat pearson_
↳functional.nii structural.nii"
>>> res = automask.run()
```

Inputs:

```
[Mandatory]
in_file1: (an existing file name)
    Filename of the first image
    argument: ``%s``, position: -2
neighborhood: (a tuple of the form: ('SPHERE' or 'RHDD' or 'TOHD', a
    float) or a tuple of the form: ('RECT', a tuple of the form: (a
    float, a float, a float)))
    The region around each voxel that will be extracted for the
    statistics calculation. Possible regions are: 'SPHERE', 'RHDD'
    (rhombic dodecahedron), 'TOHD' (truncated octahedron) with a given
    radius in mm or 'RECT' (rectangular block) with dimensions to
    specify in mm.
    argument: ``-nbhd '%s(%s)''``
stat: (a list of items which are 'pearson' or 'spearman' or
    'quadrant' or 'mutinfo' or 'normuti' or 'jointent' or 'hellinger'
    or 'crU' or 'crM' or 'crA' or 'L2slope' or 'L1slope' or 'num' or
    'ALL')
    statistics to compute. Possible names are : * pearson = Pearson
    correlation coefficient * spearman = Spearman correlation
    coefficient * quadrant = Quadrant correlation coefficient * mutinfo
    = Mutual Information * normuti = Normalized Mutual Information *
    jointent = Joint entropy * hellinger= Hellinger metric * crU =
    Correlation ratio (Unsymmetric) * crM = Correlation ratio
    (symmetrized by Multiplication) * crA = Correlation ratio
    (symmetrized by Addition) * L2slope = slope of least-squares (L2)
    linear regression of the data from dataset1 vs. the dataset2 (i.e.,
    d2 = a + b*d1 ==> this is 'b') * L1slope = slope of least-absolute-
    sum (L1) linear regression of the data from dataset1 vs. the
    dataset2 * num = number of the values in the region: with the use of
    -mask or -automask, the size of the region around any given voxel
    will vary; this option lets you map that size. * ALL = all of the
    above, in that orderMore than one option can be used.
    argument: ``-stat %s...``
in_file2: (an existing file name)
    Filename of the second image
    argument: ``%s``, position: -1

[Optional]
weight_file: (a file name)
    File name of an image to use as a weight. Only applies to 'pearson'
    statistics.
    argument: ``-weight %s``
```

(continues on next page)

(continued from previous page)

```

        mutually_exclusive: automask
out_file: (a file name)
        Output dataset.
        argument: ``-prefix %s``, position: 0
num_threads: (an integer (int or long), nipy default value: 1)
        set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
mask_file: (a file name)
        mask image file name. Voxels NOT in the mask will not be used in the
        neighborhood of any voxel. Also, a voxel NOT in the mask will have
        its statistic(s) computed as zero (0).
        argument: ``-mask %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
automask: (a boolean)
        Compute the mask as in program 3dAutomask.
        argument: ``-automask``
        mutually_exclusive: weight_file

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:

None None

51.5.19 Localstat[Link to code](#)

Wraps the executable command 3dLocalstat.

3dLocalstat - computes statistics at each voxel, based on a local neighborhood of that voxel.

For complete details, see the [3dLocalstat Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> localstat = afni.Localstat()
>>> localstat.inputs.in_file = 'functional.nii'
>>> localstat.inputs.mask_file = 'skeleton_mask.nii.gz'
>>> localstat.inputs.neighborhood = ('SPHERE', 45)
>>> localstat.inputs.stat = 'mean'
>>> localstat.inputs.nonmask = True
>>> localstat.inputs.outputtype = 'NIFTI_GZ'
>>> localstat.cmdline
"3dLocalstat -prefix functional_localstat.nii -mask skeleton_mask.nii.gz -nbhd
↪ 'SPHERE(45.0)' -use_nonmask -stat mean functional.nii"
>>> res = localstat.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input dataset
        argument: ``%s``, position: -1
stat: (a list of items which are 'mean' or 'stdev' or 'var' or 'cvar'
      or 'median' or 'MAD' or 'min' or 'max' or 'absmax' or 'num' or
      'sum' or 'FWHM' or 'FWHMbar' or 'rank' or 'frank' or 'P2skew' or
      'ALL' or 'mMP2s' or 'mmMP2s' or a tuple of the form: ('perc', a
      tuple of the form: (a float, a float, a float)))
statistics to compute. Possible names are :
  * mean = average of the values
  * stdev = standard deviation
  * var = variance (stdev*stdev)
  * cvar = coefficient of variation = stdev/fabs(mean)
  * median = median of the values
  * MAD = median absolute deviation
  * min = minimum
  * max = maximum
  * absmax = maximum of the absolute values
  * num = number of the values in the region:
    with the use of -mask or -automask, the size of the region around
    any given voxel will vary; this option lets you map that size. It
    may be useful if you plan to compute a t-statistic (say) from the
    mean and stdev outputs.
  * sum = sum of the values in the region
  * FWHM = compute (like 3dFWHM) image smoothness inside each voxel's
    neighborhood. Results are in 3 sub-bricks: FWHMx, FWHMy, and FWHMz.
    Places where an output is -1 are locations where the FWHM value
    could not be computed (e.g., outside the mask).
  * FWHMbar= Compute just the average of the 3 FWHM values (normally
    would NOT do this with FWHM also).
  * perc:P0:P1:Pstep =
    Compute percentiles between P0 and P1 with a step of Pstep.
    Default P1 is equal to P0 and default P2 = 1
  * rank = rank of the voxel's intensity
  * frank = rank / number of voxels in neighborhood
  * P2skew = Pearson's second skewness coefficient 3 * (mean -
    median) / stdev
  * ALL = all of the above, in that order (except for FWHMbar and
    perc).
  * mMP2s = Exactly the same output as: median, MAD, P2skew, but a
    little faster
  * mmMP2s = Exactly the same output as: mean, median, MAD, P2skew
    More than one option can be used.
argument: ``-stat %s...``
neighborhood: (a tuple of the form: ('SPHERE' or 'RHDD' or 'TOHD', a
      float) or a tuple of the form: ('RECT', a tuple of the form: (a
      float, a float, a float)))
The region around each voxel that will be extracted for the
statistics calculation. Possible regions are: 'SPHERE', 'RHDD'
(rhombic dodecahedron), 'TOHD' (truncated octahedron) with a given
radius in mm or 'RECT' (rectangular block) with dimensions to
specify in mm.
argument: ``-nbhd '%s(%s)''``

[Optional]
grid_rmode: ('NN' or 'Li' or 'Cu' or 'Bk')

```

(continues on next page)

(continued from previous page)

```

Interpolant to use when resampling the output with
thereduce_restore_grid option. The resampling method string RESAM
should come from the set {'NN', 'Li', 'Cu', 'BK'}. These stand for
'Nearest Neighbor', 'Linear', 'Cubic', and 'Blocky' interpolation,
respectively.
argument: ``-grid_rmode %s``
requires: reduce_restore_grid
reduce_restore_grid: (a float or a tuple of the form: (a float, a
float, a float))
Like reduce_grid, but also resample output back to inputgrid.
argument: ``-reduce_restore_grid %s``
mutually_exclusive: reduce_max_vox, reduce_grid
out_file: (a file name)
Output dataset.
argument: ``-prefix %s``, position: 0
num_threads: (an integer (int or long), nipy default value: 1)
set number of threads
reduce_max_vox: (a float)
Like reduce_restore_grid, but automatically set Rx Ry Rz sothat the
computation grid is at a resolution of nbhd/MAX_VOXvoxels.
argument: ``-reduce_max_vox %s``
mutually_exclusive: reduce_restore_grid, reduce_grid
nonmask: (a boolean)
Voxels not in the mask WILL have their local statistics computed
from all voxels in their neighborhood that ARE in the mask.
* For instance, this option can be used to compute the average
local white matter time series, even at non-WM voxels.
argument: ``-use_nonmask``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipy default value: {})
Environment variables
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
AFNI output filetype
mask_file: (a file name)
Mask image file name. Voxels NOT in the mask will not be used in the
neighborhood of any voxel. Also, a voxel NOT in the mask will have
its statistic(s) computed as zero (0) unless the parameter 'nonmask'
is set to true.
argument: ``-mask %s``
overwrite: (a boolean)
overwrite output file if it already exists
argument: ``-overwrite``
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
automask: (a boolean)
Compute the mask as in program 3dAutomask.
argument: ``-automask``
quiet: (a boolean)
Stop the highly informative progress reports.
argument: ``-quiet``
reduce_grid: (a float or a tuple of the form: (a float, a float, a
float))
Compute output on a grid that is reduced by the specified factors.
If a single value is passed, output is resampled to the specified
isotropic grid. Otherwise, the 3 inputs describe the reduction in

```

(continues on next page)

(continued from previous page)

the X, Y, and Z directions. This option speeds up computations at the expense of resolution. It should only be used when the nbhd is quite large with respect to the input's resolution, and the resultant stats are expected to be smooth.

argument: ``-reduce_grid %s``

mutually_exclusive: reduce_restore_grid, reduce_max_vox

Outputs:

out_file: (an existing file name)
output file

References:

None None

51.5.20 MaskTool

[Link to code](#)

Wraps the executable 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
argument: ``-input %s``, position: -1

[Optional]

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.
argument: ``-count``, position: 2

union: (a boolean)
union, this means -frac 0
argument: ``-union``

out_file: (a file name)
output image file name
argument: ``-prefix %s``

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.
argument: ``-fill_holes``

(continues on next page)

(continued from previous page)

```

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
    argument: ``-dilate_inputs %s``
datum: ('byte' or 'short' or 'float')
    specify data type for output. Valid types are 'byte', 'short' and
    'float'.
    argument: ``-datum %s``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
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
    argument: ``-frac %s``
inter: (a boolean)
    intersection, this means -frac 1.0
    argument: ``-inter``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
dilate_results: (a unicode string)
    dilate and/or erode combined mask at the given levels.
    argument: ``-dilate_results %s``
verbose: (an integer (int or long))
    specify verbosity level, for 0 to 3
    argument: ``-verb %s``
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}.
    argument: ``-fill_dirs %s``
requires: fill_holes

```

Outputs:

```

out_file: (an existing file name)
    mask file

```

References:

None None

51.5.21 Merge[Link to code](#)

Wraps the executable 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)
           argument: ``%s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
blurfwhm: (an integer (int or long))
           FWHM blur value (mm)
           argument: ``-lblur_fwhm %d``
doall: (a boolean)
        apply options to all sub-bricks in dataset
        argument: ``-doall``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
              set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
out_file: (a file name)
           output image file name
           argument: ``-prefix %s``
```

Outputs:

```
out_file: (an existing file name)
           output file
```

References:

None None

51.5.22 Notes

[Link to code](#)

Wraps the executable command 3dNotes.

A program to add, delete, and show notes for AFNI datasets.

For complete details, see the [3dNotes Documentation](#).

Examples

```
>>> from nipyype.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
         argument: ``%s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
ses: (a boolean)
     print to stdout the expanded notes
     argument: ``-ses``
delete: (an integer (int or long))
        delete note number num
        argument: ``-d %d``
out_file: (a file name)
          output image file name
          argument: ``%s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
rep_history: (a unicode string)
            note with which to replace history
            argument: ``-HH "%s"``
            mutually_exclusive: add_history
add: (a unicode string)
     note to add
     argument: ``-a "%s"``
num_threads: (an integer (int or long), nipyype default value: 1)
             set number of threads
environ: (a dictionary with keys which are a 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
add_history: (a unicode string)
            note to add to history
            argument: ``-h "%s"``
            mutually_exclusive: rep_history
```

Outputs:

```
out_file: (an existing file name)
          output file
```

51.5.23 NwarpAdjust

[Link to code](#)

Wraps the executable command 3dNwarpAdjust.

This program takes as input a bunch of 3D warps, averages them, and computes the inverse of this average warp. It then composes each input warp with this inverse average to ‘adjust’ the set of warps. Optionally, it can also read in a set of 1-brick datasets corresponding to the input warps, and warp each of them, and average those. For complete details, see the [3dNwarpAdjust Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> adjust = afni.NwarpAdjust()
>>> adjust.inputs.warps = ['func2anat_InverseWarp.nii.gz', 'func2anat_InverseWarp.
↳nii.gz', 'func2anat_InverseWarp.nii.gz', 'func2anat_InverseWarp.nii.gz',
↳'func2anat_InverseWarp.nii.gz']
>>> adjust.cmdline
'3dNwarpAdjust -nwarp func2anat_InverseWarp.nii.gz func2anat_InverseWarp.nii.gz_
↳func2anat_InverseWarp.nii.gz func2anat_InverseWarp.nii.gz func2anat_InverseWarp.
↳nii.gz'
>>> res = adjust.run()
```

Inputs:

```
[Mandatory]
warps: (a list of at least 5 items which are an existing file name)
      List of input 3D warp datasets
      argument: ``-nwarp %s``

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
           AFNI output filetype
out_file: (a file name)
          Output mean dataset, only needed if in_files are also given. The
          output dataset will be on the common grid shared by the source
          datasets.
          argument: ``-prefix %s``
          requires: in_files
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
in_files: (a list of at least 5 items which are an existing file
          name)
          List of input 3D datasets to be warped by the adjusted warp
          datasets. There must be exactly as many of these datasets as there
          are input warps.
          argument: ``-source %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file
```

51.5.24 NwarpApply

[Link to code](#)

Wraps the executable command 3dNwarpApply.

Program to apply a nonlinear 3D warp saved from 3dQwarp (or 3dNwarpCat, etc.) to a 3D dataset, to produce a warped version of the source dataset.

For complete details, see the [3dNwarpApply Documentation](#).

Examples

```
>>> from nipy.interfaces import afni
>>> nwarp = afni.NwarpApply()
>>> nwarp.inputs.in_file = 'Fred+orig'
>>> nwarp.inputs.master = 'NWARP'
>>> nwarp.inputs.warp = "'Fred_WARP+tlrc Fred.Xaff12.1D'"
>>> nwarp.cmdline
"3dNwarpApply -source Fred+orig -interp wsinc5 -master NWARP -prefix Fred+orig_
↳Nwarp -nwarp 'Fred_WARP+tlrc Fred.Xaff12.1D'"
>>> res = nwarp.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name or a list of items which are an
         existing file name)
         the name of the dataset to be warped can be multiple datasets
         argument: ``-source %s``
warp: (a string)
      the name of the warp dataset. multiple warps can be concatenated
      (make sure they exist)
      argument: ``-nwarp %s``

[Optional]
ainterp: ('NN' or 'nearestneighbour' or 'nearestneighbor' or 'linear'
         or 'trilinear' or 'cubic' or 'tricubic' or 'quintic' or
         'triquintic' or 'wsinc5')
         specify a different interpolation method than might be used for the
         warp
         argument: ``-ainterp %s``
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
master: (a file name)
        the name of the master dataset, which defines the output grid
        argument: ``-master %s``
verb: (a boolean)
      be extra verbose :)
      argument: ``-verb``
      mutually_exclusive: quiet
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
interp: ('wsinc5' or 'NN' or 'nearestneighbour' or 'nearestneighbor'
        or 'linear' or 'trilinear' or 'cubic' or 'tricubic' or 'quintic'
        or 'triquintic', nipy default value: wsinc5)
        defines interpolation method to use during warp
        argument: ``-interp %s``
```

(continues on next page)

(continued from previous page)

```

short: (a boolean)
    Write output dataset using 16-bit short integers, rather than the
    usual 32-bit floats.
    argument: ``-short``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
quiet: (a boolean)
    don't be verbose :(
    argument: ``-quiet``
    mutually_exclusive: verb
inv_warp: (a boolean)
    After the warp specified in '-nwarp' is computed, invert it
    argument: ``-iwarp``

```

Outputs:

```

out_file: (an existing file name)
    output file

```

51.5.25 NwarpCat

[Link to code](#)

Wraps the executable command 3dNwarpCat.

Catenates (composes) 3D warps defined on a grid, OR via a matrix.

Note:

- All transformations are from DICOM xyz (in mm) to DICOM xyz.
- Matrix warps are in files that end in '.1D' or in '.txt'. A matrix warp file should have 12 numbers in it, as output (for example), by '3dAllineate -1Dmatrix_save'.
- Nonlinear warps are in dataset files (AFNI .HEAD/.BRIK or NIfTI .nii) with 3 sub-bricks giving the DICOM order xyz grid displacements in mm.
- If all the input warps are matrices, then the output is a matrix and will be written to the file 'prefix.aff12.1D'. Unless the prefix already contains the string '.1D', in which case the filename is just the prefix.
- If 'prefix' is just 'stdout', then the output matrix is written to standard output. In any of these cases, the output format is 12 numbers in one row.
- If any of the input warps are datasets, they must all be defined on the same 3D grid! And of course, then the output will be a dataset on the same grid. However, you can expand the grid using the '-expad' option.
- The order of operations in the final (output) warp is, for the case of 3 input warps:

$$\text{OUTPUT}(x) = \text{warp3}(\text{warp2}(\text{warp1}(x)))$$
 That is, warp1 is applied first, then warp2, et cetera. The 3D x coordinates are taken from each grid location in the first dataset defined on a grid.

For complete details, see the [3dNwarpCat Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> nwarpcat = afni.NwarpCat()
>>> nwarpcat.inputs.in_files = ['Q25_warp+tlrc.HEAD', ('IDENT', 'structural.nii')]
>>> nwarpcat.inputs.out_file = 'Fred_total_WARP'
>>> nwarpcat.cmdline
"3dNwarpCat -interp wsinc5 -prefix Fred_total_WARP Q25_warp+tlrc.HEAD
↪ 'IDENT(structural.nii)'"

```

(continues on next page)

(continued from previous page)

```
>>> res = nwarpcat.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are a file name or a tuple of the
          form: ('IDENT' or 'INV' or 'SQRT' or 'SQRTINV', a file name))
          list of tuples of 3D warps and associated functions
          argument: ``%s``, position: -1

[Optional]
space: (a string)
       string to attach to the output dataset as its atlas space marker.
       argument: ``-space %s``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
inv_warp: (a boolean)
          invert the final warp before output
          argument: ``-iwarp``
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
expad: (an integer (int or long))
       Pad the nonlinear warps by the given number of voxels voxels in all
       directions. The warp displacements are extended by linear
       extrapolation from the faces of the input grid..
       argument: ``-expad %d``
verb: (a boolean)
      be verbose
      argument: ``-verb``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
interp: ('wsinc5' or 'linear' or 'quintic', nipy default value:
        wsinc5)
        specify a different interpolation method than might be used for the
        warp
        argument: ``-interp %s``
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.5.26 OneDToolPy[Link to code](#)Wraps the executable command `1d_tool.py`.

This program is meant to read/manipulate/write/diagnose 1D datasets. Input can be specified using AFNI sub-brick[**]/time{ } selectors.**

```
>>> from nipy.interfaces import afni
>>> odt = afni.OneDToolPy()
>>> odt.inputs.in_file = 'f1.1D'
>>> odt.inputs.set_nruns = 3
>>> odt.inputs.demean = True
>>> odt.inputs.out_file = 'motion_dmean.1D'
>>> odt.cmdline
'python2 ...1d_tool.py -demean -infile f1.1D -write motion_dmean.1D -set_nruns 3'
>>> res = odt.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to OneDTool
         argument: ``-infile %s``

[Optional]
py27_path: (an existing file name or 'python2', nipy default value:
            python2)
show_trs_uncensored: ('comma' or 'space' or 'encoded' or 'verbose')
                     display a list of TRs which were not censored in the specified style
                     argument: ``-show_trs_uncensored %s``
out_file: (a file name)
          write the current 1D data to FILE
          argument: ``-write %s``
          mutually_exclusive: show_cormat_warnings
censor_motion: (a tuple of the form: (a float, a file name))
               Tuple of motion limit and outfile prefix. need to also set set_nruns
               -r set_run_lengths
               argument: ``-censor_motion %f %s``
show_trs_run: (an integer (int or long))
              restrict -show_trs_[un]censored to the given 1-based run
              argument: ``-show_trs_run %d``
show_cormat_warnings: (a file name)
                     Write cormat warnings to a file
                     argument: ``-show_cormat_warnings |& tee %s``, position: -1
                     mutually_exclusive: out_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', nipy default value: {})
          Environment variables
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
derivative: (a boolean)
            take the temporal derivative of each vector (done as first backward
            difference)
            argument: ``-derivative``
censor_prev_TR: (a boolean)
                for each censored TR, also censor previous
                argument: ``-censor_prev_TR``
show_indices_interest: (a boolean)
                       display column indices for regs of interest
                       argument: ``-show_indices_interest``
args: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

Additional parameters to the command
argument: ``%s``
set_nruns: (an integer (int or long))
    treat the input data as if it has nruns
    argument: ``-set_nruns %d``
show_censor_count: (a boolean)
    display the total number of censored TRs Note : if input is a valid
    xmat.1D dataset, then the count will come from the header. Otherwise
    the input is assumed to be a binary censorfile, and zeros are simply
    counted.
    argument: ``-show_censor_count``
demean: (a boolean)
    demean each run (new mean of each run = 0.0)
    argument: ``-demean``

```

Outputs:

```

out_file: (a file name)
    output of 1D_tool.py

```

References:

None None

51.5.27 ReHo[Link to code](#)

Wraps the executable command 3dReHo.

Compute regional homogeneity for a given neighbourhood.1, based on a local neighborhood of that voxel.

For complete details, see the [3dReHo Documenta**tion**](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> reho = afni.ReHo()
>>> reho.inputs.in_file = 'functional.nii'
>>> reho.inputs.out_file = 'reho.nii.gz'
>>> reho.inputs.neighborhood = 'vertices'
>>> reho.cmdline
'3dReHo -prefix reho.nii.gz -inset functional.nii -nneigh 27'
>>> res = reho.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    input dataset
    argument: ``-inset %s``, position: 1

[Optional]
label_set: (an existing file name)
    a set of ROIs, each labelled with distinct integers. ReHo will then
    be calculated per ROI.
    argument: ``-in_rois %s``
mask_file: (a file name)
    Mask within which ReHo should be calculated voxelwise

```

(continues on next page)

(continued from previous page)

```

        argument: ``-mask %s``
overwrite: (a boolean)
    overwrite output file if it already exists
    argument: ``-overwrite``
out_file: (a file name)
    Output dataset.
    argument: ``-prefix %s``, position: 0
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
chi_sq: (a boolean)
    Output the Friedman chi-squared value in addition to the Kendall's
    W. This option is currently compatible only with the AFNI
    (BRIK/HEAD) output type; the chi-squared value will be the second
    sub-brick of the output dataset.
    argument: ``-chi_sq``
neighborhood: ('faces' or 'edges' or 'vertices')
    voxels in neighborhood. can be: * faces (for voxel and 6 facewise
    neighbors, only),
    * edges (for voxel and 18 face- and edge-wise neighbors),
    * vertices (for voxel and 26 face-, edge-, and node-wise neighbors).
    argument: ``-nneigh %s``
mutually_exclusive: sphere, ellipsoid
ellipsoid: (a tuple of the form: (a float, a float, a float))
    Tuple indicating the x, y, and z radius of an ellipsoid defining the
    neighbourhood of each voxel.
    The 'hood is then made according to the following relation:  $(i/A)^2 + (j/B)^2 + (k/C)^2 \leq 1$ .
    which will have approx.  $V=4*PI*A*B*C/3$ . The impetus for this freedom
    was for use with data having anisotropic voxel edge lengths.
    argument: ``-neigh_X %s -neigh_Y %s -neigh_Z %s``
mutually_exclusive: sphere, neighborhood
sphere: (a float)
    for additional voxelwise neighborhood control, the radius R of a
    desired neighborhood can be put in; R is a floating point number,
    and must be >1. Examples of the numbers of voxels in a given radius
    are as follows (you can roughly approximate with the ol'
     $4*PI*(R^3)/3$  thing):
    R=2.0 -> V=33,
    R=2.3 -> V=57,
    R=2.9 -> V=93,
    R=3.1 -> V=123,
    R=3.9 -> V=251,
    R=4.5 -> V=389,
    R=6.1 -> V=949,
    but you can choose most any value.
    argument: ``-neigh_RAD %s``
mutually_exclusive: neighborhood, ellipsoid
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

out_vals: (a file name)
    Table of labelwise regional homogeneity values

```

(continues on next page)

(continued from previous page)

```
out_file: (an existing file name)
          Voxelwise regional homogeneity map
```

51.5.28 Refit

[Link to code](#)

Wraps the executable 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()
```

```
>>> refit_2 = afni.Refit()
>>> refit_2.inputs.in_file = 'structural.nii'
>>> refit_2.inputs.atrfloat = ("IJK_TO_DICOM_REAL", "1 0.2 0 0 -0.2 1 0 0 0 0 1 0
↪")
>>> refit_2.cmdline
"3drefit -atrfloat IJK_TO_DICOM_REAL '1 0.2 0 0 -0.2 1 0 0 0 0 1 0' structural.nii
↪"
>>> res = refit_2.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input file to 3drefit
         argument: ``%s``, position: -1

[Optional]
duporigin_file: (an existing file name)
                Copies the xorigin, yorigin, and zorigin values from the header of
                the given dataset
                argument: ``-duporigin %s``
deoblique: (a boolean)
            replace current transformation matrix with cardinal matrix
            argument: ``-deoblique``
yorigin: (a unicode string)
          y distance for edge voxel offset
          argument: ``-yorigin %s``
nosaveatr: (a boolean)
            Opposite of -saveatr
            argument: ``-nosaveatr``
xdel: (a float)
       new x voxel dimension in mm
       argument: ``-xdel %f``
ydel: (a float)
       new y voxel dimension in mm
```

(continues on next page)

(continued from previous page)

```

        argument: ``-ydel %f``
zorigin: (a unicode string)
        z distance for edge voxel offset
        argument: ``-zorigin %s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
saveatr: (a boolean)
        (default) Copy the attributes that are known to AFNI into the
        dset->dblk structure thereby forcing changes to known attributes to
        be present in the output. This option only makes sense with
        -atrcopy.
        argument: ``-saveatr``
space: ('TLRC' or 'MNI' or 'ORIG')
        Associates the dataset with a specific template type, e.g. TLRC,
        MNI, ORIG
        argument: ``-space %s``
xorigin: (a unicode string)
        x distance for edge voxel offset
        argument: ``-xorigin %s``
atrcopy: (a tuple of the form: (a file name, a unicode string))
        Copy AFNI header attribute from the given file into the header of
        the dataset(s) being modified. For more information on AFNI header
        attributes, see documentation file README.attributes. More than one
        '-atrcopy' option can be used. For AFNI advanced users only. Do NOT
        use -atrcopy or -atrstring with other modification options. See also
        -copyaux.
        argument: ``-atrcopy %s %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
atrstring: (a tuple of the form: (a unicode string, a unicode
        string))
        Copy the last given string into the dataset(s) being modified,
        giving it the attribute name given by the last string. To be safe,
        the last string should be in quotes.
        argument: ``-atrstring %s %s``
atrfloat: (a tuple of the form: (a unicode string, a unicode string))
        Create or modify floating point attributes. The input values may be
        specified as a single string in quotes or as a 1D filename or
        string, example '1 0.2 0 0 -0.2 1 0 0 0 0 1 0' or flipZ.1D or
        '1D:1,0.2,2@0,-0.2,1,2@0,2@0,1,0'
        argument: ``-atrfloat %s %s``
atrint: (a tuple of the form: (a unicode string, a unicode string))
        Create or modify integer attributes. The input values may be
        specified as a single string in quotes or as a 1D filename or
        string, example '1 0 0 0 0 1 0 0 0 0 1 0' or flipZ.1D or
        '1D:1,0,2@0,-0,1,2@0,2@0,1,0'
        argument: ``-atrint %s %s``
xyzscale: (a float)
        Scale the size of the dataset voxels by the given factor
        argument: ``-xyzscale %f``
zdel: (a float)
        new z voxel dimension in mm
        argument: ``-zdel %f``

```

Outputs:

```
out_file: (an existing file name)
          output file
```

51.5.29 Resample

[Link to code](#)

Wraps the executable 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
         argument: ``-inset %s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
out_file: (a file name)
          output image file name
          argument: ``-prefix %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
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.
               argument: ``-rmode %s``
num_threads: (an integer (int or long), nipy default value: 1)
             set number of threads
master: (a file name)
        align dataset grid to a reference file
        argument: ``-master %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
orientation: (a unicode string)
            new orientation code
            argument: ``-orient %s``
voxel_size: (a tuple of the form: (a float, a float, a float))
            resample to new dx, dy and dz
            argument: ``-dxyz %f %f %f``
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.5.30 TCat

[Link to code](#)

Wraps the executable 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 Documenta**tion**](#).

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
          argument: `` %s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
out_file: (a file name)
           output image file name
           argument: ``-prefix %s``
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.
      argument: ``-rlt%s``, position: 1
verbose: (a boolean)
          Print out some verbose output as the program
          argument: ``-verb``
num_threads: (an integer (int or long), nipy default value: 1)
              set number of threads
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
```

(continues on next page)

(continued from previous page)

```

of class 'str' and with values which are a bytes or None or a
value of class 'str', nipy default value: {})
Environment variables

```

Outputs:

```

out_file: (an existing file name)
          output file

```

References:

None None

51.5.31 TCatSubBrick[Link to code](#)

Wraps the executable command 3dTcat.

Hopefully a temporary function to allow sub-brick selection until afni file management is improved.

For complete details, see the [3dTcat Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> tcsb = afni.TCatSubBrick()
>>> tcsb.inputs.in_files = [('functional.nii', "{2..$}"), ('functional2.nii', "
↳{2..$}")]
>>> tcsb.inputs.out_file= 'functional_tcat.nii'
>>> tcsb.inputs.rlt = '+'
>>> tcsb.cmdline
"3dTcat -rlt+ -prefix functional_tcat.nii functional.nii'{2..$}' functional2.nii'
↳{2..$}' "
>>> res = tcsb.run()

```

Inputs:

```

[Mandatory]
in_files: (a list of items which are a tuple of the form: (an
            existing file name, a unicode string))
            List of tuples of file names and subbrick selectors as strings. Don't
            forget to protect the single quotes in the subbrick selectorso the
            contents are protected from the command line interpreter.
            argument: ``%s%s ...``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
out_file: (a file name)
            output image file name
            argument: ``-prefix %s``
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.
            argument: ``-rlt%s``, position: 1

```

(continues on next page)

(continued from previous page)

```

num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.5.32 TStat[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -1

[Optional]
mask: (an existing file name)
    mask file
    argument: ``-mask %s``
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
options: (a unicode string)
    selected statistical output
    argument: ``%s``
out_file: (a file name)
    output image file name
    argument: ``-prefix %s``

```

(continues on next page)

(continued from previous page)

```

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

out_file: (an existing file name)
    output file

```

References:

None None

51.5.33 To3D[Link to code](#)Wraps the executable command `to3d`.Create a 3D dataset from 2D image files using AFNI `to3d` commandFor 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
    argument: ``%s/*.dcm``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
datatype: ('short' or 'float' or 'byte' or 'complex')
    set output file datatype
    argument: ``-datum %s``
assumemosaic: (a boolean)
    assume that Siemens image is mosaic
    argument: ``-assume_dicom_mosaic``
out_file: (a file name)
    output image file name

```

(continues on next page)

(continued from previous page)

```

        argument: ``-prefix %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
num_threads: (an integer (int or long), nipyee default value: 1)
        set number of threads
funcparams: (a unicode string)
        parameters for functional data
        argument: ``-time:zt %s alt+z2``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipyee default value: {})
        Environment variables
skipoutliers: (a boolean)
        skip the outliers check
        argument: ``-skip_outliers``
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
        argument: ``-%s``

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:

None None

51.5.34 Undump[Link to code](#)

Wraps the executable command 3dUndump.

3dUndump - Assembles a 3D dataset from an ASCII list of coordinates and (optionally) values.

The input file(s) are ASCII files, with one voxel specification per line. A voxel specification is 3 numbers (-ijk or -xyz coordinates), with an optional 4th number giving the voxel value. For example: 1 2 3 2 1 5 5.3 6.2 3.7 // this line illustrates a comment

The first line puts a voxel (with value given by '-dval') at point (1,2,3). The second line puts a voxel (with value 5) at point (3,2,1). The third line puts a voxel (with value given by '-dval') at point (5.3,6.2,3.7). If -ijk is in effect, and fractional coordinates are given, they will be rounded to the nearest integers; for example, the third line would be equivalent to (i,j,k) = (5,6,4).

For complete details, see the [3dUndump Documentation](#).**Examples**

```

>>> from nipyee.interfaces import afni
>>> unndump = afni.Undump()
>>> unndump.inputs.in_file = 'structural.nii'
>>> unndump.inputs.out_file = 'structural_undumped.nii'
>>> unndump.cmdline
'3dUndump -prefix structural_undumped.nii -master structural.nii'
>>> res = unndump.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input file to 3dUndump, whose geometry will determinethe geometry of
        the output
        argument: ``-master %s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
        AFNI output filetype
out_file: (a file name)
        output image file name
        argument: ``-prefix %s``
fill_value: (a float)
        value, used for each voxel in the output dataset that is NOT listed
        in the input file
        argument: ``-fval %f``
num_threads: (an integer (int or long), nipy default value: 1)
        set number of threads
orient: (a tuple of the form: ('R' or 'L', 'A' or 'P', 'I' or 'S'))
        Specifies the coordinate order used by -xyz. The code must be 3
        letters, one each from the pairs {R,L} {A,P} {I,S}. The first letter
        gives the orientation of the x-axis, the second the orientation of
        the y-axis, the third the z-axis: R = right-to-left L = left-to-
        right A = anterior-to-posterior P = posterior-to-anterior I =
        inferior-to-superior S = superior-to-inferior If -orient isn't used,
        then the coordinate order of the -master (in_file) dataset is used
        to interpret (x,y,z) inputs.
        argument: ``-orient %s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
default_value: (a float)
        default value stored in each input voxel that does not have a value
        supplied in the input file
        argument: ``-dval %f``
head_only: (a boolean)
        create only the .HEAD file which gets exploited by the AFNI matlab
        library function New_HEAD.m
        argument: ``-head_only``
datatype: ('short' or 'float' or 'byte')
        set output file datatype
        argument: ``-datum %s``
mask_file: (a file name)
        mask image file name. Only voxels that are nonzero in the mask can
        be set.
        argument: ``-mask %s``
coordinates_specification: ('ijk' or 'xyz')
        Coordinates in the input file as index triples (i, j, k) or spatial
        coordinates (x, y, z) in mm
        argument: ``-%s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
srاد: (a float)
        radius in mm of the sphere that will be filled about each input

```

(continues on next page)

(continued from previous page)

```
(x,y,z) or (i,j,k) voxel. If the radius is not given, or is 0, then
each input data line sets the value in only one voxel.
argument: ``-srad %f``
```

Outputs:

```
out_file: (an existing file name)
          assembled file
```

References:

None None

51.5.35 Unifize[Link to code](#)

Wraps the executable 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
         argument: ``-input %s``, position: -1

[Optional]
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
```

(continues on next page)

(continued from previous page)

```

        contain the number of masks that it is set in.
        argument: ``-T2``
gm: (a boolean)
    Also scale to unifize 'gray matter' = lower intensity voxels (to aid
    in registering images from different scanners).
    argument: ``-GM``
out_file: (a file name)
    output image file name
    argument: ``-prefix %s``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
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.
    argument: ``-EPI``
    mutually_exclusive: gm
    requires: no_duplo, t2
rbt: (a tuple of the form: (a float, a float, a float))
    Option for AFNI experts only. Specify the 3 parameters for the
    algorithm:
    R = radius; same as given by option '-Urad', [default=18.3]
    b = bottom percentile of normalizing data range, [default=70.0]
    r = top percentile of normalizing data range, [default=80.0]
    argument: ``-rbt %f %f %f``
t2_up: (a float)
    Option for AFNI experts only. Set the upper percentile point used for
    T2-T1 inversion. Allowed to be anything between 90 and 100
    (inclusive), with default to 98.5 (for no good reason).
    argument: ``-T2up %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
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
cl_frac: (a float)
    Option for AFNI experts only. Set the automask 'clip level fraction'.
    Must be between 0.1 and 0.9. A small fraction means to make the
    initial threshold for clipping (a la 3dClipLevel) smaller, which
    will tend to make the mask larger. [default=0.1]
    argument: ``-clfrac %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
scale_file: (a file name)
    output file name to save the scale factor used at each voxel
    argument: ``-ssave %s``
quiet: (a boolean)
    Don't print the progress messages.
    argument: ``-quiet``
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.

```

(continues on next page)

(continued from previous page)

```

        argument: ``-Urad %s``
no_duplo: (a boolean)
    Do NOT use the 'duplo down' step; this can be useful for lower
    resolution datasets.
    argument: ``-noduplo``

```

Outputs:

```

out_file: (an existing file name)
    unified file
scale_file: (a file name)
    scale factor file

```

References:

None None

51.5.36 ZCutUp[Link to code](#)

Wraps the executable command 3dZcutup.

Cut z-slices from a volume using AFNI 3dZcutup command

For complete details, see the [3dZcutup Documentation](#).**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
    argument: ``%s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
keep: (a unicode string)
    slice range to keep in output
    argument: ``-keep %s``
out_file: (a file name)
    output image file name
    argument: ``-prefix %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads

```

(continues on next page)

(continued from previous page)

```
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

51.5.37 Zcat[Link to code](#)

Wraps the executable command 3dZcat.

Copies an image of one type to an image of the same or different type using 3dZcat command

For complete details, see the [3dZcat Documenta**tion**](#).**Examples**

```
>>> from nipy.interfaces import afni
>>> zcat = afni.Zcat()
>>> zcat.inputs.in_files = ['functional2.nii', 'functional3.nii']
>>> zcat.inputs.out_file = 'cat_functional.nii'
>>> zcat.cmdline
'3dZcat -prefix cat_functional.nii functional2.nii functional3.nii'
>>> res = zcat.run()
```

Inputs:

```
[Mandatory]
in_files: (a list of items which are an existing file name)
          argument: ``%s``, position: -1

[Optional]
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
            AFNI output filetype
datum: ('byte' or 'short' or 'float')
        specify data type for output. Valid types are 'byte', 'short' and
        'float'.
        argument: ``-datum %s``
nscale: (a boolean)
        Don't do any scaling on output to byte or short datasets. This may
        be especially useful when operating on mask datasets whose output
        values are only 0's and 1's.
        argument: ``-nscale``
        mutually_exclusive: fscale
out_file: (a file name)
          output dataset prefix name (default 'zcat')
          argument: ``-prefix %s``
fscale: (a boolean)
        Force scaling of the output to the maximum integer range. This only
        has effect if the output datum is byte or short (either forced or
```

(continues on next page)

(continued from previous page)

```

        defaulted). This option is sometimes necessary to eliminate
        unpleasant truncation artifacts.
        argument: ``-fscale``
        mutually_exclusive: nscale
num_threads: (an integer (int or long), nipy default value: 1)
        set number of threads
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
verb: (a boolean)
        print out some verbosity as the program proceeds.
        argument: ``-verb``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

out_file: (an existing file name)
        output file

```

References:

None None

51.5.38 Zeropad[Link to code](#)

Wraps the executable command 3dZeropad.

Adds planes of zeros to a dataset (i.e., pads it out).

For complete details, see the [3dZeropad Documentation](#).**Examples**

```

>>> from nipy.interfaces import afni
>>> zeropad = afni.Zeropad()
>>> zeropad.inputs.in_files = 'functional.nii'
>>> zeropad.inputs.out_file = 'pad_functional.nii'
>>> zeropad.inputs.I = 10
>>> zeropad.inputs.S = 10
>>> zeropad.inputs.A = 10
>>> zeropad.inputs.P = 10
>>> zeropad.inputs.R = 10
>>> zeropad.inputs.L = 10
>>> zeropad.cmdline
'3dZeropad -A 10 -I 10 -L 10 -P 10 -R 10 -S 10 -prefix pad_functional.nii_
↳functional.nii'
>>> res = zeropad.run()

```

Inputs:

```

[Mandatory]
in_files: (an existing file name)
        input dataset
        argument: ``%s``, position: -1

```

(continues on next page)

(continued from previous page)

```

[Optional]
out_file: (a file name)
    output dataset prefix name (default 'zeropad')
    argument: ``-prefix %s``
A: (an integer (int or long))
    adds 'n' planes of zero at the Anterior edge
    argument: ``-A %i``
    mutually_exclusive: master
P: (an integer (int or long))
    adds 'n' planes of zero at the Posterior edge
    argument: ``-P %i``
    mutually_exclusive: master
num_threads: (an integer (int or long), nipy default value: 1)
    set number of threads
L: (an integer (int or long))
    adds 'n' planes of zero at the Left edge
    argument: ``-L %i``
    mutually_exclusive: master
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
I: (an integer (int or long))
    adds 'n' planes of zero at the Inferior edge
    argument: ``-I %i``
    mutually_exclusive: master
outputtype: ('AFNI' or 'NIFTI_GZ' or 'NIFTI')
    AFNI output filetype
IS: (an integer (int or long))
    specify that planes should be added or cut symmetrically to make the
    resulting volume haveN slices in the inferior-superior direction
    argument: ``-IS %i``
    mutually_exclusive: master
R: (an integer (int or long))
    adds 'n' planes of zero at the Right edge
    argument: ``-R %i``
    mutually_exclusive: master
AP: (an integer (int or long))
    specify that planes should be added or cut symmetrically to make the
    resulting volume haveN slices in the anterior-posterior direction
    argument: ``-AP %i``
    mutually_exclusive: master
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
mm: (a boolean)
    pad counts 'n' are in mm instead of slices, where each 'n' is an
    integer and at least 'n' mm of slices will be added/removed; e.g., n
    = 3 and slice thickness = 2.5 mm ==> 2 slices added
    argument: ``-mm``
    mutually_exclusive: master
master: (a file name)
    match the volume described in dataset 'mset', where mset must have
    the same orientation and grid spacing as dataset to be padded. the
    goal of -master is to make the output dataset from 3dZeropad match
    the spatial 'extents' of mset by adding or subtracting slices as

```

(continues on next page)

(continued from previous page)

```
needed. You can't use -I, -S, ..., or -mm with -master
argument: ``-master %s``
mutually_exclusive: I, S, A, P, L, R, z, RL, AP, IS, mm
S: (an integer (int or long))
  adds 'n' planes of zero at the Superior edge
  argument: ``-S %i``
  mutually_exclusive: master
RL: (an integer (int or long))
  specify that planes should be added or cut symmetrically to make the
  resulting volume haveN slices in the right-left direction
  argument: ``-RL %i``
  mutually_exclusive: master
z: (an integer (int or long))
  adds 'n' planes of zero on EACH of the dataset z-axis (slice-
  direction) faces
  argument: ``-z %i``
  mutually_exclusive: master
```

Outputs:

```
out_file: (an existing file name)
          output file
```

References:

None None

52.1 interfaces.ants.legacy

52.1.1 GenWarpFields

[Link to code](#)

Wraps the executable command `antsIntroduction.sh`.

Inputs:

```
[Mandatory]
input_image: (an existing file name)
    input image to warp to template
    argument: ``-i %s``
reference_image: (an existing file name)
    template file to warp to
    argument: ``-r %s``

[Optional]
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)
    argument: ``-s %s``
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
    argument: ``-m %s``
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
bias_field_correction: (a boolean)
    Applies bias field correction to moving image
    argument: ``-n 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
```

(continues on next page)

(continued from previous page)

```

out_prefix: (a unicode string, nipyte default value: ants_)
    Prefix that is prepended to all output files (default = ants_)
    argument: ``-o %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
force_proceed: (a boolean)
    force script to proceed even if headers may be incompatible
    argument: ``-f 1``
transformation_model: ('GR' or 'EL' or 'SY' or 'S2' or 'EX' or 'DD'
    or 'RI' or 'RA', nipyte 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
    argument: ``-t %s``
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)
    argument: ``-l``
quality_check: (a boolean)
    Perform a quality check of the result
    argument: ``-q 1``
dimension: (3 or 2, nipyte default value: 3)
    image dimension (2 or 3)
    argument: ``-d %d``, position: 1

```

Outputs:

```

affine_transformation: (an existing file name)
    affine (prefix_Affine.txt)
input_file: (an existing file name)
    input image (prefix_repaired.nii)
warp_field: (an existing file name)
    warp field (prefix_Warp.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)

```

52.1.2 antsIntroduction[Link to code](#)Wraps the executable command `antsIntroduction.sh`.

Uses ANTS to generate matrices to warp data from one space to another.

Examples

```

>>> from nipyte.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'

```

(continues on next page)

(continued from previous page)

Inputs:

```

[Mandatory]
input_image: (an existing file name)
    input image to warp to template
    argument: ``-i %s``
reference_image: (an existing file name)
    template file to warp to
    argument: ``-r %s``

[Optional]
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)
    argument: ``-s %s``
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
    argument: ``-m %s``
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
bias_field_correction: (a boolean)
    Applies bias field correction to moving image
    argument: ``-n 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
out_prefix: (a unicode string, nipy default value: ants_)
    Prefix that is prepended to all output files (default = ants_)
    argument: ``-o %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
force_proceed: (a boolean)
    force script to proceed even if headers may be incompatible
    argument: ``-f 1``
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
    argument: ``-t %s``
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)
    argument: ``-l``
quality_check: (a boolean)
    Perform a quality check of the result
    argument: ``-q 1``
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)

```

(continues on next page)

(continued from previous page)

```
argument: ``-d %d``, position: 1
```

Outputs:

```
affine_transformation: (an existing file name)
    affine (prefix_Affine.txt)
input_file: (an existing file name)
    input image (prefix_repaired.nii)
warp_field: (an existing file name)
    warp field (prefix_Warp.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)
```

52.1.3 buildtemplateparallel

[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -1

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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)
    argument: ``-s %s``
num_cores: (an integer (int or long))
    Requires parallelization = 2 (PEEXEC). Sets number of cpu cores to
    use
    argument: ``-j %d``
    requires: parallelization
max_iterations: (a list of items which are an integer (int or long))
```

(continues on next page)

(continued from previous page)

```

        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
        argument: ``-m %s``
num_threads: (an integer (int or long), nipy default value: 1)
        Number of ITK threads to use
iteration_limit: (an integer (int or long), nipy default value: 4)
        iterations of template construction
        argument: ``-i %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
out_prefix: (a unicode string, nipy default value: antsTMPL_)
        Prefix that is prepended to all output files (default = antsTMPL_)
        argument: ``-o %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.
bias_field_correction: (a boolean)
        Applies bias field correction to moving image
        argument: ``-n 1``
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
        argument: ``-t %s``
gradient_step_size: (a float)
        smaller magnitude results in more cautious steps (default = .25)
        argument: ``-g %f``
rigid_body_registration: (a boolean)
        registers inputs before creating template (useful if no initial
        template available)
        argument: ``-r 1``
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
        argument: ``-c %d``
dimension: (3 or 2 or 4, nipy default value: 3)
        image dimension (2, 3 or 4)
        argument: ``-d %d``, position: 1

```

Outputs:

```

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)
final_template_file: (an existing file name)
        final ANTS template
template_files: (a list of items which are an existing file name)
        Templates from different stages of iteration

```

52.2 interfaces.ants.registration

52.2.1 MeasureImageSimilarity

[Link to code](#)

Wraps the executable command MeasureImageSimilarity.

Examples

```
>>> from nipyte.interfaces.ants import MeasureImageSimilarity
>>> sim = MeasureImageSimilarity()
>>> sim.inputs.dimension = 3
>>> sim.inputs.metric = 'MI'
>>> sim.inputs.fixed_image = 'T1.nii'
>>> sim.inputs.moving_image = 'resting.nii'
>>> sim.inputs.metric_weight = 1.0
>>> sim.inputs.radius_or_number_of_bins = 5
>>> sim.inputs.sampling_strategy = 'Regular'
>>> sim.inputs.sampling_percentage = 1.0
>>> sim.inputs.fixed_image_mask = 'mask.nii'
>>> sim.inputs.moving_image_mask = 'mask.nii.gz'
>>> sim.cmdline
'MeasureImageSimilarity --dimensionality 3 --masks ["mask.nii","mask.nii.gz"] --
↪metric MI["T1.nii","resting.nii",1.0,5,Regular,1.0]'
```

Inputs:

```
[Mandatory]
moving_image: (an existing file name)
    Image to apply transformation to (generally a coregistered
    functional)
sampling_percentage: (0.0 <= a floating point number <= 1.0)
    Percentage of points accessible to the sampling strategy over which
    to optimize the metric.
    requires: metric
metric: ('CC' or 'MI' or 'Mattes' or 'MeanSquares' or 'Demons' or
    'GC')
    argument: ``%s``
fixed_image: (an existing file name)
    Image to which the moving image is warped
radius_or_number_of_bins: (an integer (int or long))
    The number of bins in each stage for the MI and Mattes metric, or
    the radius for other metrics
    requires: metric

[Optional]
sampling_strategy: ('None' or 'Regular' or 'Random', nipyte default
    value: None)
    Manner of choosing point set over which to optimize the metric.
    Defaults to "None" (i.e. a dense sampling of one sample per voxel).
    requires: metric
num_threads: (an integer (int or long), nipyte default value: 1)
    Number of ITK threads to use
moving_image_mask: (an existing file name)
    mask used to limit metric sampling region of the moving image
    requires: fixed_image_mask
environ: (a dictionary with keys which are a bytes or None or a value
```

(continues on next page)

(continued from previous page)

```

    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
metric_weight: (a float, nipy default value: 1.0)
    The "metricWeight" variable is not used.
    requires: metric
fixed_image_mask: (an existing file name)
    mask used to limit metric sampling region of the fixed image
    argument: ``%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
dimension: (2 or 3 or 4)
    Dimensionality of the fixed/moving image pair
    argument: ``--dimensionality %d``, position: 1

```

Outputs:

```

similarity: (a float)

```

52.2.2 Registration[Link to code](#)

Wraps the executable command `antsRegistration`.

ANTs Registration command for registration of images

`antsRegistration` registers a `moving_image` to a `fixed_image`, using a predefined (sequence of) cost function(s) and transformation operations. The cost function is defined using one or more ‘metrics’, specifically local cross-correlation (CC), Mean Squares (MeanSquares), Demons (Demons), global correlation (GC), or Mutual Information (Mattes or MI).

ANTs can use both linear (Translation, Rigid, Affine, CompositeAffine, or Translation) and non-linear transformations (BSpline, GaussianDisplacementField, TimeVaryingVelocityField, TimeVaryingBSplineVelocityField, SyN, BSplineSyN, Exponential, or BSplineExponential). Usually, registration is done in multiple *stages*. For example first an Affine, then a Rigid, and ultimately a non-linear (Syn)-transformation.

`antsRegistration` can be initialized using one or more transforms from `moving_image` to `fixed_image` with the `initial_moving_transform`-input. For example, when you already have a warfield that corrects for geometrical distortions in an EPI (functional) image, that you want to apply before an Affine registration to a structural image. You could put this transform into ‘`initial_moving_transform`’.

The Registration-interface can output the resulting transform(s) that map `moving_image` to `fixed_image` in a single file as a `composite_transform` (if `write_composite_transform` is set to `True`), or a list of transforms as `forwards_transforms`. It can also output inverse transforms (from `fixed_image` to `moving_image`) in a similar fashion using `inverse_composite_transform`. Note that the order of `forward_transforms` is in ‘natural’ order: the first element should be applied first, the last element should be applied last.

Note, however, that ANTs tools always apply lists of transformations in reverse order (the last transformation in the list is applied first). Therefore, if the output `forward_transforms` is a list, one can not directly feed it into, for example, `ants.ApplyTransforms`. To make `ants.ApplyTransforms` apply the transformations in the same order as `ants.Registration`, you have to provide the list of transformations in reverse order from `forward_transforms`. `reverse_forward_transforms` outputs `forward_transforms` in reverse order and can be used for this purpose. Note also that, because `composite_transform` is always a single file, this output is preferred for most use-cases.

More information can be found in the [ANTs manual](#).

See below for some useful examples.

Examples

Set up a Registration node with some default settings. This Node registers ‘fixed1.nii’ to ‘moving1.nii’ by first fitting a linear ‘Affine’ transformation, and then a non-linear ‘SyN’ transformation, both using the Mutual Information-cost metric.

The registration is initialized by first applying the (linear) transform trans.mat.

```
>>> import copy, pprint
>>> from nipyte.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()
```

Same as reg1, but first invert the initial transform (‘trans.mat’) before applying it.

```
>>> 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'
```

(continued from previous page)

```
>>> reg1.run()
```

Clip extremely high intensity data points using `winsorize_upper_quantile`. All data points higher than the 0.975 quantile are set to the value of the 0.975 quantile.

```
>>> 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'
```

Clip extremely low intensity data points using `winsorize_lower_quantile`. All data points lower than the 0.025 quantile are set to the original value at the 0.025 quantile.

```
>>> 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'
```

Use float instead of double for computations (saves memory usage)

```
>>> 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'
```

Force to use double instead of float for computations (more precision and memory usage).

```

>>> 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'

```

'collapse_output_transforms' can be used to put all transformation in a single 'composite_transform'- file. Note that forward_transforms will now be an empty list.

```

>>> # 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': '...data/output_Composite.h5',
 'elapsed_time': <undefined>,
 'forward_invert_flags': [],
 'forward_transforms': [],
 'inverse_composite_transform': '...data/output_InverseComposite.h5',
 'inverse_warped_image': <undefined>,
 'metric_value': <undefined>,
 'reverse_invert_flags': [],
 'reverse_transforms': [],
 'save_state': '...data/trans.mat',
 'warped_image': '...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>,
 'elapsed_time': <undefined>,

```

(continues on next page)

(continued from previous page)

```

'forward_invert_flags': [False, False],
'forward_transforms': ['...data/output_0GenericAffine.mat',
'...data/output_1Warp.nii.gz'],
'inverse_composite_transform': <undefined>,
'inverse_warped_image': <undefined>,
'metric_value': <undefined>,
'reverse_invert_flags': [True, False],
'reverse_transforms': ['...data/output_0GenericAffine.mat',      '...data/output_
↳1InverseWarp.nii.gz'],
'save_state': '...data/trans.mat',
'warped_image': '...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'

```

One can use multiple similarity metrics in a single registration stage. The Node below first performs a linear registration using only the Mutual Information (‘Mattes’)-metric. In a second stage, it performs a non-linear registration (‘Syn’) using both a Mutual Information and a local cross-correlation (‘CC’)-metric. Both metrics are weighted equally (‘metric_weight’ is .5 for both). The Mutual Information- metric uses 32 bins. The local cross-correlations (correlations between every voxel’s neighborhoods) is computed with a radius of 4.

```

>>> # 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'

```

ANTS Registration can also use multiple modalities to perform the registration. Here it is assumed that fixed1.nii and fixed2.nii are in the same space, and so are moving1.nii and moving2.nii. First, a linear registration is performed matching fixed1.nii to moving1.nii, then a non-linear registration is performed to match fixed2.nii to moving2.nii, starting from the transformation of the first step.

```
>>> # 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'
```

Different methods can be used for the interpolation when applying transformations.

```
>>> # 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'
```

BSplineSyN non-linear registration with custom parameters.

```
>>> # Test Extended Transform Parameters
>>> reg8 = copy.deepcopy(reg)
>>> reg8.inputs.transforms = ['Affine', 'BSplineSyN']
```

(continues on next page)

(continued from previous page)

```
>>> 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'
```

Mask the fixed image in the second stage of the registration (but not the first).

```
>>> # Test masking
>>> reg9 = copy.deepcopy(reg)
>>> reg9.inputs.fixed_image_masks = ['NULL', 'fixed1.nii']
>>> reg9.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 --masks [ NULL, NULL ] --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 --masks [ fixed1.nii, NULL ] --winsorize-
↪image-intensities [ 0.0, 1.0 ] --write-composite-transform 1'
```

Here we use both a warpfild and a linear transformation, before registration commences. Note that the first transformation that needs to be applied ('ants_Warp.nii.gz') is last in the list of 'initial_moving_transform'.

```
>>> # Test initialization with multiple transforms matrices (e.g., unwarp and
↪affine transform)
>>> reg10 = copy.deepcopy(reg)
>>> reg10.inputs.initial_moving_transform = ['func_to_struct.mat', 'ants_Warp.nii.
↪gz']
>>> reg10.inputs.invert_initial_moving_transform = [False, False]
>>> reg10.cmdline
'antsRegistration --collapse-output-transforms 0 --dimensionality 3 --initial-
↪moving-transform [ func_to_struct.mat, 0 ] [ ants_Warp.nii.gz, 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'
```

Inputs:

```
[Mandatory]
metric_weight: (a list of items which are a float or a list of items
```

(continues on next page)

(continued from previous page)

```

        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
fixed_image: (a list of items which are an existing file name)
        Image to which the moving_image should be transformed(usually a
        structural image)
smoothing_sigmas: (a list of items which are a list of items which
        are a float)
shrink_factors: (a list of items which are a list of items which are
        an integer (int or long))
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')
        argument: ``%s``
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.
moving_image: (a list of items which are an existing file name)
        Image that will be registered to the space of fixed_image. This is
        the image on which the transformations will be applied to

[Optional]
radius_bins_item_trait: (an integer (int or long), nipy default
        value: 5)
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)
metric_weight_stage_trait: (a float or a list of items which are a
        float)
use_histogram_matching: (a boolean or a list of items which are a
        boolean, nipy default value: True)
        Histogram match the images before registration.
restrict_deformation: (a list of items which are a list of items
        which are 0 or 1)
        This option allows the user to restrict the optimization of the
        displacement field, translation, rigid or affine transform on a per-
        component basis. For example, if one wants to limit the deformation
        or rotation of 3-D volume to the first two dimensions, this is
        possible by specifying a weight vector of '1x1x0' for a deformation
        field or '1x1x0x1x1x0' for a rigid transformation. Low-dimensional
        restriction only works if there are no preceding transformations.
radius_bins_stage_trait: (an integer (int or long) or a list of items
        which are an integer (int or long))
metric_weight_item_trait: (a float, nipy default value: 1.0)
moving_image_masks: (a list of items which are an existing file name
        or 'NULL')
        Masks used to limit metric sampling region of the moving image,
        defined per registration stage(Use "NULL" to omit a mask at a given

```

(continues on next page)

(continued from previous page)

```

    stage)
    mutually_exclusive: moving_image_mask
sigma_units: (a list of items which are 'mm' or 'vox')
    units for smoothing sigmas
    requires: smoothing_sigmas
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).
    argument: ``--initialize-transforms-per-stage %d``
interpolation_parameters: (a tuple of the form: (an integer (int or
    long)) or a tuple of the form: (a float, a float))
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
collapse_output_transforms: (a boolean, nipy default value: True)
    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.
    argument: ``--collapse-output-transforms %d``
use_estimate_learning_rate_once: (a list of items which are a
    boolean)
invert_initial_moving_transform: (a list of items which are a
    boolean)
    One boolean or a list of booleans that indicate whether the
    inverse(s) of the transform(s) defined in initial_moving_transform
    should be used.
    mutually_exclusive: initial_moving_transform_com
    requires: initial_moving_transform
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
initial_moving_transform: (a list of items which are an existing file
    name)
    A transform or a list of transforms that should be applied before the
    registration begins. Note that, when a list is given, the
    transformations are applied in reverse order.
    argument: ``%s``
    mutually_exclusive: initial_moving_transform_com
write_composite_transform: (a boolean, nipy default value: False)
    argument: ``--write-composite-transform %d``
moving_image_mask: (an existing file name)
    mask used to limit metric sampling region of the moving image in all
    stages
    mutually_exclusive: moving_image_masks
    requires: fixed_image_mask
winsorize_upper_quantile: (0.0 <= a floating point number <= 1.0,
    nipy default value: 1.0)
    The Upper quantile to clip image ranges
    argument: ``%s``
convergence_threshold: (a list of at least 1 items which are a float,

```

(continues on next page)

(continued from previous page)

```

        nipy default value: [1e-06])
    requires: number_of_iterations
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')
winsorize_lower_quantile: (0.0 <= a floating point number <= 1.0,
    nipy default value: 0.0)
    The Lower quantile to clip image ranges
    argument: ``%s``
float: (a boolean)
    Use float instead of double for computations.
    argument: ``--float %d``
save_state: (a file name)
    Filename for saving the internal restorable state of the
    registration
    argument: ``--save-state %s``
fixed_image_masks: (a list of items which are an existing file name
    or 'NULL')
    Masks used to limit metric sampling region of the fixed image,
    defined per registration stage(Use "NULL" to omit a mask at a given
    stage)
    mutually_exclusive: fixed_image_mask
initial_moving_transform_com: (0 or 1 or 2)
    Align the moving_image nad fixed_image befor registration usingthe
    geometric center of the images (=0), the image intensities (=1),or
    the origin of the images (=2)
    argument: ``%s``
    mutually_exclusive: initial_moving_transform
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)
number_of_iterations: (a list of items which are a list of items
    which are an integer (int or long))
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
output_warped_image: (a boolean or a file name)
fixed_image_mask: (an existing file name)
    Mask used to limit metric sampling region of the fixed imagein all
    stages
    argument: ``%s``
    mutually_exclusive: fixed_image_masks
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
interpolation: ('Linear' or 'NearestNeighbor' or 'CosineWindowedSinc'
    or 'WelchWindowedSinc' or 'HammingWindowedSinc' or
    'LanczosWindowedSinc' or 'BSpline' or 'MultiLabel' or 'Gaussian',
    nipy default value: Linear)
    argument: ``%s``
convergence_window_size: (a list of at least 1 items which are an
    integer (int or long), nipy default value: [10])
    requires: convergence_threshold

```

(continues on next page)

(continued from previous page)

```

num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
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)
output_transform_prefix: (a unicode string, nipy default value:
    transform)
    argument: ``%s``
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    argument: ``--dimensionality %d``
output_inverse_warped_image: (a boolean or a file name)
    requires: output_warped_image
verbose: (a boolean, nipy default value: False)
    argument: ``-v``
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)))
restore_state: (an existing file name)
    Filename for restoring the internal restorable state of the
    registration
    argument: ``--restore-state %s``
metric_item_trait: ('CC' or 'MeanSquares' or 'Demons' or 'GC' or 'MI'
    or 'Mattes')
sampling_percentage_item_trait: (0.0 <= a floating point number <=
    1.0 or None)

```

Outputs:

```

forward_invert_flags: (a list of items which are a boolean)
    List of flags corresponding to the forward transforms
inverse_composite_transform: (a file name)
    Inverse composite transform file
warped_image: (a file name)
    Outputs warped image
save_state: (a file name)
    The saved registration state to be restored
reverse_transforms: (a list of items which are an existing file name)
    List of output transforms for reverse registration
elapsed_time: (a float)
    the total elapsed time as reported by ANTs
reverse_invert_flags: (a list of items which are a boolean)
    List of flags corresponding to the reverse transforms
forward_transforms: (a list of items which are an existing file name)
    List of output transforms for forward registration
inverse_warped_image: (a file name)
    Outputs the inverse of the warped image
metric_value: (a float)
    the final value of metric
composite_transform: (an existing file name)
    Composite transform file

```

52.2.3 RegistrationSynQuick

[Link to code](#)

Wraps the executable command `antsRegistrationSynQuick.sh`.

Registration using a symmetric image normalization method (SyN). You can read more in Avants et al.; Med Image Anal., 2008 (<https://www.ncbi.nlm.nih.gov/pubmed/17659998>).

Examples

```
>>> from nipy.interfaces.ants import RegistrationSynQuick
>>> reg = RegistrationSynQuick()
>>> reg.inputs.fixed_image = 'fixed1.nii'
>>> reg.inputs.moving_image = 'moving1.nii'
>>> reg.inputs.num_threads = 2
>>> reg.cmdline
'antsRegistrationSynQuick.sh -d 3 -f fixed1.nii -r 32 -m moving1.nii -n 2 -o_
↳transform -p d -s 26 -t s'
>>> reg.run()
```

example for multiple images

```
>>> from nipy.interfaces.ants import RegistrationSynQuick
>>> reg = RegistrationSynQuick()
>>> reg.inputs.fixed_image = ['fixed1.nii', 'fixed2.nii']
>>> reg.inputs.moving_image = ['moving1.nii', 'moving2.nii']
>>> reg.inputs.num_threads = 2
>>> reg.cmdline
'antsRegistrationSynQuick.sh -d 3 -f fixed1.nii -f fixed2.nii -r 32 -m moving1.
↳nii -m moving2.nii -n 2 -o transform -p d -s 26 -t s'
>>> reg.run()
```

Inputs:

```
[Mandatory]
moving_image: (a list of items which are an existing file name)
    Moving image or target image
    argument: ``-m %s...``
fixed_image: (a list of items which are an existing file name)
    Fixed image or source image or reference image
    argument: ``-f %s...``

[Optional]
output_prefix: (a unicode string, nipy default value: transform)
    A prefix that is prepended to all output files
    argument: ``-o %s``
histogram_bins: (an integer (int or long), nipy default value: 32)
    histogram bins for mutual information in SyN stage (default = 32)
    argument: ``-r %d``
use_histogram_matching: (a boolean)
    use histogram matching
    argument: ``-j %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
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    argument: ``-d %d``
```

(continues on next page)

(continued from previous page)

```

precision_type: ('double' or 'float', nipy default value: double)
    precision type (default = double)
    argument: ``-p %s``
transform_type: ('s' or 't' or 'r' or 'a' or 'sr' or 'b' or 'br',
    nipy default value: s)
    transform type
    t: translation
    r: rigid
    a: rigid + affine
    s: rigid + affine + deformable syn (default)
    sr: rigid + deformable syn
    b: rigid + affine + deformable b-spline syn
    br: rigid + deformable b-spline syn
    argument: ``-t %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
spline_distance: (an integer (int or long), nipy default value: 26)
    spline distance for deformable B-spline SyN transform (default = 26)
    argument: ``-s %d``
num_threads: (an integer (int or long), nipy default value: 1)
    Number of threads (default = 1)
    argument: ``-n %d``

```

Outputs:

```

forward_warp_field: (an existing file name)
    Forward warp field
out_matrix: (an existing file name)
    Affine matrix
inverse_warped_image: (an existing file name)
    Inverse warped image
inverse_warp_field: (an existing file name)
    Inverse warp field
warped_image: (an existing file name)
    Warped image

```

52.3 interfaces.ants.resampling

52.3.1 ApplyTransforms

[Link to code](#)Wraps the executable 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.input_image = 'moving1.nii'
>>> at.inputs.reference_image = 'fixed1.nii'
>>> at.inputs.transforms = 'identity'
>>> at.cmdline
'antsApplyTransforms --default-value 0 --float 0 --input moving1.nii --
↳ interpolation Linear --output moving1_trans.nii --reference-image fixed1.nii
↳ identity'

```

(continues on next page)

(continued from previous page)

```

>>> 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 --float 0 --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]
>>> at1.cmdline
'antsApplyTransforms --default-value 0 --dimensionality 3 --float 0 --input_
↪moving1.nii --interpolation BSpline[ 5 ] --output deformed_moving1.nii --
↪reference-image fixed1.nii --transform [ ants_Warp.nii.gz, 0 ] --transform [_
↪trans.mat, 0 ]'

```

Inputs:

[Mandatory]

transforms: (a list of items which are an existing file name or 'identity')
 transform files: will be applied in reverse order. For example, the last specified transform will be applied first.
 argument: ``%s``

input_image: (an existing file name)
 image to apply transformation to (generally a coregistered functional)
 argument: ``--input %s``

reference_image: (an existing file name)
 reference image space that you wish to warp INTO
 argument: ``--reference-image %s``

[Optional]

interpolation: ('Linear' or 'NearestNeighbor' or 'CosineWindowedSinc' or 'WelchWindowedSinc' or 'HammingWindowedSinc' or 'LanczosWindowedSinc' or 'MultiLabel' or 'Gaussian' or 'BSpline',
 nipyte default value: Linear)
 argument: ``%s``

print_out_composite_warp_file: (a boolean)
 output a composite warp file instead of a transformed image

(continues on next page)

(continued from previous page)

```

        requires: output_image
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
default_value: (a float, nipy default value: 0.0)
    argument: ``--default-value %g``
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
input_image_type: (0 or 1 or 2 or 3)
    Option specifying the input image type of scalar (default), vector,
    tensor, or time series.
    argument: ``--input-image-type %d``
invert_transform_flags: (a list of items which are a boolean)
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
interpolation_parameters: (a tuple of the form: (an integer (int or
    long)) or a tuple of the form: (a float, a float))
out_postfix: (a unicode string, nipy default value: _trans)
    Postfix that is appended to all output files (default = _trans)
float: (a boolean, nipy default value: False)
    Use float instead of double for computations.
    argument: ``--float %d``
output_image: (a unicode string)
    output file name
    argument: ``--output %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.
    argument: ``--dimensionality %d``

```

Outputs:

```

output_image: (an existing file name)
    Warped image

```

52.3.2 ApplyTransformsToPoints[Link to code](#)Wraps the executable 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 ]'

```

(continues on next page)

(continued from previous page)

Inputs:

```
[Mandatory]
transforms: (a list of items which are an existing file name)
            transforms that will be applied to the points
            argument: ``%s``
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 space required by
            antsApplyTransformsToPoints try creating/drawing a simple label
            volume with only one voxel set to 1 and all others set to 0. Write
            down the voxel coordinates. Then use ImageMaths LabelStats to find
            out what coordinates for this voxel antsApplyTransformsToPoints is
            expecting.
            argument: ``--input %s``

[Optional]
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.
            argument: ``--dimensionality %d``
args: (a unicode string)
            Additional parameters to the command
            argument: ``%s``
invert_transform_flags: (a list of items which are a boolean)
            list indicating if a transform should be reversed
output_file: (a unicode string)
            Name of the output CSV file
            argument: ``--output %s``
num_threads: (an integer (int or long), nipy default value: 1)
            Number of ITK threads to use
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
```

Outputs:

```
output_file: (an existing file name)
            csv file with transformed coordinates
```

52.3.3 WarpImageMultiTransform[Link to code](#)

Wraps the executable command WarpImageMultiTransform.

Warp an image from one space to another

Examples

```
>>> from nipy.interfaces.ants import WarpImageMultiTransform
>>> wimt = WarpImageMultiTransform()
```

(continues on next page)

(continued from previous page)

```
>>> 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] # this will invert the 1st Affine file:
↪'func2anat_coreg_Affine.txt'
>>> 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]
transformation_series: (a list of items which are an existing file
                        name)
                        transformation file(s) to be applied
                        argument: ``%s``, position: -1
input_image: (a file name)
              image to apply transformation to (generally a coregistered
              functional)
              argument: ``%s``, position: 2

[Optional]
dimension: (3 or 2, nipy default value: 3)
            image dimension (2 or 3)
            argument: ``%d``, position: 1
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
              Number of ITK threads to use
reslice_by_header: (a boolean)
                   Uses orientation matrix and origin encoded in reference image file
                   header. Not typically used with additional transforms
                   argument: ``--reslice-by-header``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
reference_image: (a file name)
                 reference image space that you wish to warp INTO
                 argument: ``-R %s``
                 mutually_exclusive: tightest_box
tightest_box: (a boolean)
               computes tightest bounding box (overridden by reference_image if
               given)
               argument: ``--tightest-bounding-box``
               mutually_exclusive: reference_image
```

(continues on next page)

(continued from previous page)

```

out_postfix: (a file name, nipy default value: _wimt)
    Postfix that is prepended to all output files (default = _wimt)
    mutually_exclusive: output_image
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.
output_image: (a file name)
    name of the output warped image
    argument: ``%s``, position: 3
    mutually_exclusive: out_postfix
use_nearest: (a boolean)
    Use nearest neighbor interpolation
    argument: ``--use-NN``
use_bspline: (a boolean)
    Use 3rd order B-Spline interpolation
    argument: ``--use-BSpline``

```

Outputs:

```

output_image: (an existing file name)
    Warped image

```

52.3.4 WarpTimeSeriesImageMultiTransform[Link to code](#)

Wraps the executable command WarpTimeSeriesImageMultiTransform.

Warpes 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'

```

```

>>> 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.inputs.invert_affine = [1] # # this will invert the 1st Affine file:
↳ants_Affine.txt
>>> wtsimt.cmdline
'WarpTimeSeriesImageMultiTransform 4 resting.nii resting_wtsimt.nii -R ants_
↳deformed.nii.gz ants_Warp.nii.gz -i ants_Affine.txt'

```

Inputs:

```

[Mandatory]
transformation_series: (a list of items which are an existing file

```

(continues on next page)

(continued from previous page)

```

        name)
        transformation file(s) to be applied
        argument: ``%s``
input_image: (a file name)
        image to apply transformation to (generally a coregistered
        functional)
        argument: ``%s``

[Optional]
out_postfix: (a unicode string, nipy default value: _wtsimt)
        Postfix that is prepended to all output files (default = _wtsimt)
        argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
        Number of ITK threads to use
reslice_by_header: (a boolean)
        Uses orientation matrix and origin encoded in reference image file
        header. Not typically used with additional transforms
        argument: ``--reslice-by-header``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
reference_image: (a file name)
        reference image space that you wish to warp INTO
        argument: ``-R %s``
        mutually_exclusive: tightest_box
tightest_box: (a boolean)
        computes tightest bounding box (overridden by reference_image if
        given)
        argument: ``--tightest-bounding-box``
        mutually_exclusive: reference_image
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
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.
dimension: (4 or 3, nipy default value: 4)
        image dimension (3 or 4)
        argument: ``%d``, position: 1
use_nearest: (a boolean)
        Use nearest neighbor interpolation
        argument: ``--use-NN``
use_bspline: (a boolean)
        Use 3rd order B-Spline interpolation
        argument: ``--use-Bspline``

```

Outputs:

```

output_image: (an existing file name)
        Warped image

```

52.4 interfaces.ants.segmentation

52.4.1 AntsJointFusion

[Link to code](#)

Wraps the executable 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 = [ ['rc1s1.nii', 'rc1s2.nii'] ]
>>> antsjointfusion.inputs.atlas_segmentation_image = ['segmentation0.nii.gz']
>>> antsjointfusion.inputs.target_image = ['im1.nii']
>>> antsjointfusion.cmdline
"antsJointFusion -a 0.1 -g ['rc1s1.nii', 'rc1s2.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 ['rc1s1.nii', 'rc1s2.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'
```

(continues on next page)

(continued from previous page)

```
>>> 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]
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.
argument: ``-t %s``
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.
argument: ``-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.
argument: ``-l %s...``

[Optional]
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
exclusion_image_label: (a list of items which are a unicode string)
Specify a label for the exclusion region.
argument: ``-e %s``
requires: exclusion_image
beta: (a float, nipy default value: 2.0)
Exponent for mapping intensity difference to the joint error.
Default = 2.0
argument: ``-b %s``
patch_radius: (a list of items which are a value of class 'int')
Patch radius for similarity measures.Default: 2x2x2
argument: ``-p %s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipy default value: {})
Environment variables
search_radius: (a list of from 1 to 3 items which are any value,
nipy 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.
argument: ``-s %s``
out_intensity_fusion_name_format: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

Optional intensity fusion image file name format. (e.g.
"antsJointFusionIntensity_%d.nii.gz")
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_label_fusion: (a file name)
The output label fusion image.
argument: ``%s``
retain_label_posterior_images: (a boolean, nipy default value:
False)
Retain label posterior probability images. Requires atlas
segmentations to be specified. Default = false
argument: ``-r``
requires: atlas_segmentation_image
mask_image: (an existing file name)
If a mask image is specified, fusion is only performed in the mask
region.
argument: ``-x %s``
retain_atlas_voting_images: (a boolean, nipy default value: False)
Retain atlas voting images. Default = false
argument: ``-f``
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
num_threads: (an integer (int or long), nipy default value: 1)
Number of ITK threads to use
exclusion_image: (a list of items which are an existing file name)
Specify an exclusion region for the given label.
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.
argument: ``-d %d``
constrain_nonnegative: (a boolean, nipy default value: False)
Constrain solution to non-negative weights.
argument: ``-c``
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).
argument: ``-m %s``
verbose: (a boolean)
Verbose output.
argument: ``-v``
alpha: (a float, nipy default value: 0.1)
Regularization term added to matrix Mx for calculating the inverse.
Default = 0.1
argument: ``-a %s``

```

Outputs:

```

out_label_fusion: (an existing file name)
out_label_post_prob_name_format: (a unicode string)
out_intensity_fusion_name_format: (a unicode string)
out_atlas_voting_weight_name_format: (a unicode string)

```

52.4.2 Atropos

[Link to code](#)

Wraps the executable 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 nipy.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]
mask_image: (an existing file name)
    argument: ``--mask-image %s``
initialization: ('Random' or 'Otsu' or 'KMeans' or
    'PriorProbabilityImages' or 'PriorLabelImage')
    argument: ``%s``
requires: number_of_tissue_classes
number_of_tissue_classes: (an integer (int or long))
intensity_images: (a list of items which are an existing file name)
    argument: ``--intensity-image %s...``

[Optional]
maximum_number_of_icm_terations: (an integer (int or long))
    requires: icm_use_synchronous_update
prior_weighting: (a float)
save_posteriors: (a boolean)
icm_use_synchronous_update: (a boolean)
    argument: ``%s``
```

(continues on next page)

(continued from previous page)

```

out_classified_image_name: (a file name)
    argument: ``%s``
likelihood_model: (a unicode string)
    argument: ``--likelihood-model %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
prior_probability_threshold: (a float)
    requires: prior_weighting
output_posteriors_name_template: (a unicode string, nipy default
    value: POSTERIOR_%02d.nii.gz)
use_random_seed: (a boolean, nipy default value: True)
    use random seed value over constant
    argument: ``--use-random-seed %d``
posterior_formulation: (a unicode string)
    argument: ``%s``
prior_probability_images: (a list of items which are an existing file
    name)
mrf_radius: (a list of items which are an integer (int or long))
    requires: mrf_smoothing_factor
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
n_iterations: (an integer (int or long))
    argument: ``%s``
mrf_smoothing_factor: (a float)
    argument: ``%s``
use_mixture_model_proportions: (a boolean)
    requires: posterior_formulation
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
convergence_threshold: (a float)
    requires: n_iterations
dimension: (3 or 2 or 4, nipy default value: 3)
    image dimension (2, 3, or 4)
    argument: ``--image-dimensionality %d``

```

Outputs:

```

classified_image: (an existing file name)
posteriors: (a list of items which are a file name)

```

52.4.3 BrainExtraction[Link to code](#)Wraps the executable 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'

```

(continues on next page)

(continued from previous page)

```
>>> 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]
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.
    argument: ``-m %s``

brain_template: (an existing file name)
    Anatomical template created using e.g. LPBA40 data set with
    buildtemplateparallel.sh in ANTs.
    argument: ``-e %s``

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.
    argument: ``-a %s``

[Optional]
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.
    argument: ``-f %s``

num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use

environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

keep_temporary_files: (an integer (int or long))
    Keep brain extraction/segmentation warps, etc (default = 0).
    argument: ``-k %d``

out_prefix: (a unicode string, nipy default value: highres001_)
    Prefix that is prepended to all output files (default =
    highres001_)
    argument: ``-o %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.
    argument: ``-z 1``

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

use_random_seeding: (0 or 1)
    Use random number generated from system clock in Atropos (default =
    1)
    argument: ``-u %d``

use_floatingpoint_precision: (0 or 1)
```

(continues on next page)

(continued from previous page)

```

    Use floating point precision in registrations (default = 0)
    argument: ``-q %d``
image_suffix: (a unicode string, nipy default value: nii.gz)
    any of standard ITK formats, nii.gz is default
    argument: ``-s %s``
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    argument: ``-d %d``

```

Outputs:

```
BrainExtractionTemplateLaplacian: (an existing file name)
BrainExtractionPrior1Warp: (an existing file name)
BrainExtractionTmp: (an existing file name)
BrainExtractionInitialAffine: (an existing file name)
BrainExtractionPrior1InverseWarp: (an existing file name)
BrainExtractionSegmentation: (an existing file name)
    segmentation mask with CSF, GM, and WM
BrainExtractionInitialAffineMoving: (an existing file name)
BrainExtractionLaplacian: (an existing file name)
N4Corrected0: (an existing file name)
    N4 bias field corrected image
BrainExtractionPriorWarped: (an existing file name)
BrainExtractionMask: (an existing file name)
    brain extraction mask
N4Truncated0: (an existing file name)
BrainExtractionWM: (an existing file name)
    segmenration mask with only white matter
BrainExtractionInitialAffineFixed: (an existing file name)
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
BrainExtractionPrior0GenericAffine: (an existing file name)
```

52.4.4 CorticalThickness

Link to code

Wraps the executable 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',
...
'BrainSegmentationPrior02.nii.
↳ gz',
...
'BrainSegmentationPrior03.nii.
↳ gz',
```

(continues on next page)

(continued from previous page)

```

...                                     'BrainSegmentationPrior04.nii.
↪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 nipype_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 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.
    argument: ``-a %s``
brain_probability_mask: (an existing file name)
    brain probability mask in template space
    argument: ``-m %s``
brain_template: (an existing file name)
    Anatomical *intensity* template (possibly created using a population
    data set with buildtemplateparallel.sh in ANTs). This template is
    *not* skull-stripped.
    argument: ``-e %s``
segmentation_priors: (a list of items which are an existing file
    name)
    argument: ``-p %s``
t1_registration_template: (an existing file name)
    Anatomical *intensity* template (assumed to be skull-stripped). A
    common case would be where this would be the same template as
    specified in the -e option which is not skull stripped.
    argument: ``-t %s``

[Optional]
extraction_registration_mask: (an existing file name)
    Mask (defined in the template space) used during registration for
    brain extraction.
    argument: ``-f %s``
environ: (a dictionary with keys which are a 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
keep_temporary_files: (an integer (int or long))
    Keep brain extraction/segmentation warps, etc (default = 0).
    argument: ``-k %d``
out_prefix: (a unicode string, nipype default value: antsCT_)
    Prefix that is prepended to all output files (default = antsCT_)
    argument: ``-o %s``
use_random_seeding: (0 or 1)
    Use random number generated from system clock in Atropos (default =
    1)
    argument: ``-u %d``
b_spline_smoothing: (a boolean)
    Use B-spline SyN for registrations and B-spline exponential mapping

```

(continues on next page)

(continued from previous page)

```

        in DiReCT.
        argument: ``-v``
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).
    argument: ``-b %s``
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].
    argument: ``-l %s``
max_iterations: (an integer (int or long))
    ANTS registration max iterations (default = 100x100x70x20)
    argument: ``-i %d``
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
segmentation_iterations: (an integer (int or long))
    N4 -> Atropos -> N4 iterations during segmentation (default = 3)
    argument: ``-n %d``
prior_segmentation_weight: (a float)
    Atropos spatial prior *probability* weight for the segmentation
    argument: ``-w %f``
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.
    argument: ``-z 1``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
use_floatingpoint_precision: (0 or 1)
    Use floating point precision in registrations (default = 0)
    argument: ``-j %d``
image_suffix: (a unicode string, nipy default value: nii.gz)
    any of standard ITK formats, nii.gz is default
    argument: ``-s %s``
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).
    argument: ``-q 1``
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    argument: ``-d %d``
cortical_label_image: (an existing file name)
    Cortical ROI labels to use as a prior for ATITH.

```

Outputs:

```

SubjectToTemplate0GenericAffine: (an existing file name)
    Template to subject inverse affine
CorticalThicknessNormedToTemplate: (an existing file name)
    Normalized cortical thickness
CorticalThickness: (an existing file name)
    cortical thickness file
BrainExtractionMask: (an existing file name)
    brain extraction mask
SubjectToTemplateLogJacobian: (an existing file name)
    Template to subject log jacobian
SubjectToTemplate1Warp: (an existing file name)
    Template to subject inverse warp
TemplateToSubject1GenericAffine: (an existing file name)
    Template to subject affine
BrainSegmentationPosteriors: (a list of items which are an existing
    file name)
    Posterior probability images
BrainVolumes: (an existing file name)
    Brain volumes as text
BrainSegmentation: (an existing file name)
    brain segmentaion image
BrainSegmentationN4: (an existing file name)
    N4 corrected image
TemplateToSubject0Warp: (an existing file name)
    Template to subject warp
ExtractedBrainN4: (an existing file name)
    extracted brain from N4 image

```

52.4.5 DenoiseImage

[Link to code](#)

Wraps the executable 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'

```

(continues on next page)

Inputs:

```

[Mandatory]
save_noise: (a boolean, nipyte default value: False)
    True if the estimated noise should be saved to file.
    mutually_exclusive: noise_image
input_image: (an existing file name)
    A scalar image is expected as input for noise correction.
    argument: ``-i %s``

[Optional]
output_image: (a file name)
    The output consists of the noise corrected version of the input
    image.
    argument: ``-o %s``
noise_model: ('Gaussian' or 'Rician', nipyte default value: Gaussian)
    Employ a Rician or Gaussian noise model.
    argument: ``-n %s``
verbose: (a boolean)
    Verbose output.
    argument: ``-v``
noise_image: (a file name)
    Filename for the estimated noise.
shrink_factor: (an integer (int or long), nipyte 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.
    argument: ``-s %s``
num_threads: (an integer (int or long), nipyte default value: 1)
    Number of ITK threads to use
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a 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
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.
    argument: ``-d %d``

```

Outputs:

```

noise_image: (a file name)
output_image: (an existing file name)

```

52.4.6 JointFusion

[Link to code](#)

Wraps the executable command jointfusion.

Examples

```
>>> from nipy.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]
target_image: (a list of items which are an existing file name)
    Target image(s)
    argument: ``-tg %s...``
output_label_image: (a file name)
    Output fusion label map image
    argument: ``%s``, position: -1
warped_label_images: (a list of items which are an existing file
    name)
    Warped atlas segmentations
    argument: ``-l %s...``
warped_intensity_images: (a list of items which are an existing file
    name)
    Warped atlas images
    argument: ``-g %s...``
modalities: (an integer (int or long))
    Number of modalities or features
    argument: ``%d``, position: 1
dimension: (3 or 2 or 4, nipy default value: 3)
    image dimension (2, 3, or 4)
    argument: ``%d``, position: 0

[Optional]
patch_radius: (a list of items which are a value of class 'int')
    Patch radius for similarity measures, scalar or vector. Default:
    2x2x2
```

(continues on next page)

(continued from previous page)

```

        argument: ``-rp %s``
atlas_group_id: (a list of items which are a value of class 'int')
    Assign a group ID for each atlas
    argument: ``-gp %d...``
exclusion_region: (an existing file name)
    Specify an exclusion region for the given label.
    argument: ``-x %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
alpha: (a float, nipy default value: 0.0)
    Regularization term added to matrix Mx for inverse
    requires: method
beta: (an integer (int or long), nipy default value: 0)
    Exponent for mapping intensity difference to joint error
    requires: method
search_radius: (a list of items which are a value of class 'int')
    Local search radius. Default: 3x3x3
    argument: ``-rs %s``
atlas_group_weights: (a list of items which are a value of class
    'int')
    Assign the voting weights to each atlas group
    argument: ``-gpw %d...``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
method: (a unicode string, nipy 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]
    argument: ``-m %s``
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use

```

Outputs:

```
output_label_image: (an existing file name)
```

52.4.7 KellyKapowski[Link to code](#)

Wraps the executable command KellyKapowski.

Nipy 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 nipy.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.thickness_prior_estimate = 10
>>> kk.cmdline

```

(continues on next page)

(continued from previous page)

```
'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 --maximum-number-of-invert-
→displacement-field-iterations 20 --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.
    argument: ``--segmentation-image "%s"``

[Optional]
gradient_step: (a float, nipy default value: 0.025)
    Gradient step size for the optimization.
    argument: ``--gradient-step %f``
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    argument: ``--image-dimensionality %d``
warped_white_matter: (a file name)
    Filename for the warped white matter file.
number_integration_points: (an integer (int or long), nipy default
    value: 10)
    Number of compositions of the diffeomorphism per iteration.
    argument: ``--number-of-integration-points %d``
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
max_invert_displacement_field_iters: (an integer (int or long),
    nipy default value: 20)
    Maximum number of iterations for estimating the invert displacement
    field.
    argument: ``--maximum-number-of-invert-displacement-field-iterations
    %d``
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.
    argument: ``--white-matter-probability-image "%s"``
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.
    argument: ``--gray-matter-probability-image "%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
```

(continues on next page)

(continued from previous page)

```

        size) and determining the slope which is then compared with the
        convergence threshold.
        argument: ``--convergence "%s"``
thickness_prior_estimate: (a float, nipy default value: 10)
    Provides a prior constraint on the final thickness measurement in
    mm.
    argument: ``--thickness-prior-estimate %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
cortical_thickness: (a file name)
    Filename for the cortical thickness.
    argument: ``--output "%s"``
smoothing_velocity_field: (a float, nipy default value: 1.5)
    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).
    argument: ``--smoothing-velocity-field-parameter %f``
white_matter_label: (an integer (int or long), nipy default value:
    3)
    The label value for the white matter label in the
    segmentation_image.
smoothing_variance: (a float, nipy default value: 1.0)
    Defines the Gaussian smoothing of the hit and total images.
    argument: ``--smoothing-variance %f``
use_bspline_smoothing: (a boolean)
    Sets the option for B-spline smoothing of the velocity field.
    argument: ``--use-bspline-smoothing 1``
thickness_prior_image: (an existing file name)
    An image containing spatially varying prior thickness values.
    argument: ``--thickness-prior-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

52.4.8 LaplacianThickness[Link to code](#)

Wraps the executable 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'

```

(continues on next page)

(continued from previous page)

```
>>> cort_thick.cmdline
'LaplacianThickness white_matter.nii.gz gray_matter.nii.gz white_matter_thickness.
↳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
    argument: ``%s``, position: 2
input_wm: (a file name)
    white matter segmentation image
    argument: ``%s``, position: 1

[Optional]
sulcus_prior: (a boolean)
    argument: ``use-sulcus-prior``, position: 7
output_image: (a file name)
    name of output file
    argument: ``%s``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
opt_tolerance: (a float)
    argument: ``optional-laplacian-tolerance=%d``, position: 8
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
prior_thickness: (a float)
    argument: ``priorthickval=%d``, 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
smooth_param: (a float)
    argument: ``smoothparam=%d``, position: 4
dT: (a float)
    argument: ``dT=%d``, position: 6
```

Outputs:

```
output_image: (an existing file name)
    Cortical thickness
```

52.4.9 N4BiasFieldCorrection[Link to code](#)

Wraps the executable 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 nipyte.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]
save_bias: (a boolean, nipyte default value: False)
    True if the estimated bias should be saved to file.
    mutually_exclusive: bias_image
input_image: (a file name)
    input for bias correction. Negative values or values close to zero
    should be processed prior to correction
    argument: ``--input-image %s``
copy_header: (a boolean, nipyte default value: False)
    copy headers of the original image into the output (corrected) file

[Optional]
bspline_order: (an integer (int or long))
    requires: bspline_fitting_distance
shrink_factor: (an integer (int or long))
```

(continues on next page)

(continued from previous page)

```

        argument: ``--shrink-factor %d``
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
n_iterations: (a list of items which are an integer (int or long))
    argument: ``--convergence %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
weight_image: (a file name)
    image for relative weighting (e.g. probability map of the white
    matter) of voxels during the B-spline fitting.
    argument: ``--weight-image %s``
bspline_fitting_distance: (a float)
    argument: ``--bspline-fitting %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
mask_image: (a file name)
    image to specify region to perform final bias correction in
    argument: ``--mask-image %s``
output_image: (a unicode string)
    output file name
    argument: ``--output %s``
convergence_threshold: (a float)
    requires: n_iterations
dimension: (3 or 2 or 4, nipy default value: 3)
    image dimension (2, 3 or 4)
    argument: ``-d %d``
bias_image: (a file name)
    Filename for the estimated bias.

```

Outputs:

```

output_image: (an existing file name)
    Warped image
bias_image: (an existing file name)
    Estimated bias

```

52.5 interfaces.ants.utils

52.5.1 AffineInitializer

[Link to code](#)Wraps the executable 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]
moving_image: (an existing file name)
    moving image
    argument: ``%s``, position: 2
fixed_image: (an existing file name)
    reference image
    argument: ``%s``, position: 1

[Optional]
radian_fraction: (0.0 <= a floating point number <= 1.0, nipy
    default value: 0.1)
    search this arc +/- principal axes
    argument: ``%f``, position: 5
dimension: (3 or 2, nipy default value: 3)
    dimension
    argument: ``%s``, position: 0
out_file: (a file name, nipy default value: transform.mat)
    output transform file
    argument: ``%s``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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
    argument: ``%d``, position: 7
principal_axes: (a boolean, nipy default value: False)
    whether the rotation is searched around an initial principal axis
    alignment.
    argument: ``%d``, position: 6
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
search_factor: (a float, nipy default value: 15.0)
    increments (degrees) for affine search
    argument: ``%f``, 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

```

Outputs:

```

out_file: (a file name)
    output transform file

```

52.5.2 AverageAffineTransform[Link to code](#)

Wraps the executable command AverageAffineTransform.

Examples

```

>>> from nipy.interfaces.ants import AverageAffineTransform
>>> avg = AverageAffineTransform()
>>> avg.inputs.dimension = 3
>>> avg.inputs.transforms = ['trans.mat', 'func_to_struct.mat']

```

(continues on next page)

(continued from previous page)

```
>>> avg.inputs.output_affine_transform = 'MYtemplatewarp.mat'
>>> avg.cmdline
'AverageAffineTransform 3 MYtemplatewarp.mat trans.mat func_to_struct.mat'
```

Inputs:

```
[Mandatory]
transforms: (a list of items which are an existing file name)
             transforms to average
             argument: ``%s``, position: 3
output_affine_transform: (a file name)
                         Outputfname.txt: the name of the resulting transform.
                         argument: ``%s``, position: 1
dimension: (3 or 2)
            image dimension (2 or 3)
            argument: ``%d``, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
num_threads: (an integer (int or long), nipyre default value: 1)
              Number of ITK threads to use
environ: (a dictionary with keys which are a 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
```

Outputs:

```
affine_transform: (an existing file name)
                  average transform file
```

52.5.3 AveragelImages

[Link to code](#)

Wraps the executable command AverageImages.

Examples

```
>>> from nipyre.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 = ['rclsl.nii', 'rclsl.nii']
>>> avg.cmdline
'AverageImages 3 average.nii.gz 1 rclsl.nii rclsl.nii'
```

Inputs:

```
[Mandatory]
images: (a list of items which are an existing file name)
         image to apply transformation to (generally a coregistered
         functional)
         argument: ``%s``, position: 3
normalize: (a boolean)
```

(continues on next page)

(continued from previous page)

```

        Normalize: if true, the 2nd image is divided by its mean. This will
                    select the largest image to average into.
                    argument: ``%d``, position: 2
dimension: (3 or 2)
            image dimension (2 or 3)
            argument: ``%d``, position: 0

[Optional]
output_average_image: (a file name, nipy default value:
                       average.nii)
                       the name of the resulting image.
                       argument: ``%s``, position: 1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
num_threads: (an integer (int or long), nipy default value: 1)
              Number of ITK threads to use
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables

```

Outputs:

```

output_average_image: (an existing file name)
                      average image file

```

52.5.4 ComposeMultiTransform[Link to code](#)

Wraps the executable command ComposeMultiTransform.

Take a set of transformations and convert them to a single transformation matrix/warfield.

Examples

```

>>> from nipy.interfaces.ants import ComposeMultiTransform
>>> compose_transform = ComposeMultiTransform()
>>> compose_transform.inputs.dimension = 3
>>> compose_transform.inputs.transforms = ['struct_to_template.mat', 'func_to_
↳ struct.mat']
>>> compose_transform.cmdline
'ComposeMultiTransform 3 struct_to_template_composed.mat struct_to_template.mat_
↳ func_to_struct.mat '

```

Inputs:

```

[Mandatory]
transforms: (a list of items which are an existing file name)
            transforms to average
            argument: ``%s``, position: 3

[Optional]
output_transform: (a file name)
                  the name of the resulting transform.
                  argument: ``%s``, position: 1
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

        Additional parameters to the command
        argument: ``%s``
num_threads: (an integer (int or long), nipyte default value: 1)
        Number of ITK threads to use
reference_image: (a file name)
        Reference image (only necessary when output is warpfield)
        argument: ``%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', nipyte default value: {})
        Environment variables
dimension: (3 or 2, nipyte default value: 3)
        image dimension (2 or 3)
        argument: ``%d``, position: 0

```

Outputs:

```

output_transform: (an existing file name)
        Composed transform file

```

52.5.5 CreateJacobianDeterminantImage

[Link to code](#)

Wraps the executable command CreateJacobianDeterminantImage.

Examples

```

>>> from nipyte.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
        argument: ``%s``, position: 1
imageDimension: (3 or 2)
        image dimension (2 or 3)
        argument: ``%d``, position: 0
outputImage: (a file name)
        output filename
        argument: ``%s``, position: 2

[Optional]
doLogJacobian: (0 or 1)
        return the log jacobian
        argument: ``%d``, position: 3
num_threads: (an integer (int or long), nipyte default value: 1)
        Number of ITK threads to use
args: (a unicode string)
        Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
useGeometric: (0 or 1)
    return the geometric jacobian
    argument: ``%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

```

Outputs:

```

jacobian_image: (an existing file name)
    jacobian image

```

52.5.6 LabelGeometry[Link to code](#)

Wraps the executable command LabelGeometryMeasures.

Extracts geometry measures using a label file and an optional image file

Examples

```

>>> from nipy.interfaces.ants import LabelGeometry
>>> label_extract = LabelGeometry()
>>> label_extract.inputs.dimension = 3
>>> label_extract.inputs.label_image = 'atlas.nii.gz'
>>> label_extract.cmdline
'LabelGeometryMeasures 3 atlas.nii.gz [] atlas.csv'

```

```

>>> label_extract.inputs.intensity_image = 'ants_Warp.nii.gz'
>>> label_extract.cmdline
'LabelGeometryMeasures 3 atlas.nii.gz ants_Warp.nii.gz atlas.csv'

```

Inputs:

```

[Mandatory]
label_image: (a file name)
    label image to use for extracting geometry measures
    argument: ``%s``, position: 1
intensity_image: (an existing file name, nipy default value: [])
    Intensity image to extract values from. This is an optional input
    argument: ``%s``, position: 2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
output_file: (a unicode string)
    name of output file
    argument: ``%s``, position: 3
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
Environment variables

```

(continues on next page)

(continued from previous page)

```
dimension: (3 or 2, nipyte default value: 3)
    image dimension (2 or 3)
    argument: ``%d``, position: 0
```

Outputs:

```
output_file: (an existing file name)
    CSV file of geometry measures
```

52.5.7 MultiplyImages

[Link to code](#)

Wraps the executable command MultiplyImages.

Examples

```
>>> from nipyte.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]
first_input: (an existing file name)
    image 1
    argument: ``%s``, position: 1
output_product_image: (a file name)
    Outputfname.nii.gz: the name of the resulting image.
    argument: ``%s``, position: 3
second_input: (an existing file name or a float)
    image 2 or multiplication weight
    argument: ``%s``, position: 2
dimension: (3 or 2)
    image dimension (2 or 3)
    argument: ``%d``, position: 0

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long), nipyte default value: 1)
    Number of ITK threads to use
environ: (a dictionary with keys which are a 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
```

Outputs:

```
output_product_image: (an existing file name)
    average image file
```

52.6 interfaces.ants.visualization

52.6.1 ConvertScalarImageToRGB

[Link to code](#)

Wraps the executable 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]
minimum_input: (an integer (int or long))
    minimum input
    argument: ``%d``, position: 6
colormap: (a unicode string, nipy default value: )
    Possible colormaps: grey, red, green, blue, copper, jet, hsv,
    spring, summer, autumn, winter, hot, cool, overunder, custom
    argument: ``%s``, position: 4
input_image: (an existing file name)
    Main input is a 3-D grayscale image.
    argument: ``%s``, position: 1
dimension: (3 or 2, nipy default value: 3)
    image dimension (2 or 3)
    argument: ``%d``, position: 0
maximum_input: (an integer (int or long))
    maximum input
    argument: ``%d``, position: 7

[Optional]
custom_color_map_file: (a unicode string, nipy default value: none)
    custom color map file
    argument: ``%s``, position: 5
num_threads: (an integer (int or long), nipy default value: 1)
    Number of ITK threads to use
maximum_RGB_output: (an integer (int or long), nipy default value:
    255)
    argument: ``%d``, position: 9
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
minimum_RGB_output: (an integer (int or long), nipy default value:
    0)
    argument: ``%d``, position: 8
args: (a unicode string)
    Additional parameters to the command
```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
mask_image: (an existing file name, nipy default value: none)
    mask image
        argument: ``%s``, position: 3
output_image: (a unicode string, nipy default value: rgb.nii.gz)
    rgb output image
        argument: ``%s``, position: 2

```

Outputs:

```

output_image: (an existing file name)
    converted RGB image

```

52.6.2 CreateTiledMosaic

[Link to code](#)

Wraps the executable 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]
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.
    argument: ``-r %s``
input_image: (an existing file name)
    Main input is a 3-D grayscale image.
    argument: ``-i %s``

[Optional]
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.
    argument: ``-t %s``
num_threads: (an integer (int or long), nipy default value: 1)

```

(continues on next page)

(continued from previous page)

```

    Number of ITK threads to use
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
alpha_value: (a float)
    If an Rgb image is provided, render the overlay using the specified
    alpha parameter.
    argument: ``-a %.2f``
slices: (a unicode string)
    Number of slices to increment Slice1xSlice2xSlice3[numberOfSlicesToI
    ncrement,<minSlice=0>,<maxSlice=lastSlice>]
    argument: ``-s %s``
permute_axes: (a boolean)
    doPermute
    argument: ``-g``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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".
    argument: ``-p %s``
mask_image: (an existing file name)
    Specifies the ROI of the RGB voxels used.
    argument: ``-x %s``
output_image: (a unicode string, nipy default value: output.png)
    The output consists of the tiled mosaic image.
    argument: ``-o %s``
flip_slice: (a unicode string)
    flipXxflipY
    argument: ``-f %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.
    argument: ``-d %d``

```

Outputs:

```

output_image: (an existing file name)
    image file

```

53.1 interfaces.base.core

53.1.1 BaseInterface

[Link to code](#)

Implements common interface functionality.

Implements

- Initializes inputs/outputs from input_spec/output_spec
- Provides help based on input_spec and output_spec
- Checks for mandatory inputs before running an interface
- Runs an interface and returns results
- Determines which inputs should be copied or linked to cwd

This class does not implement aggregate_outputs, input_spec or output_spec. These should be defined by derived classes.

This class cannot be instantiated.

Relevant Interface attributes

input_spec points to the traitled class for the inputs output_spec points to the traitled class for the outputs
_redirect_x should be set to True when the interface requires
connecting to a \$DISPLAY (default is False).

resource_monitor if **False** prevents resource-monitoring this interface, if True monitoring will be enabled IFF the general Nipype config is set on (resource_monitor = true).

Inputs:

None

Outputs:

None

53.1.2 CommandLine

[Link to code](#)

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

Use get_traitsfree() to check all inputs set >>> pprint.pprint(cli.inputs.get_traitsfree()) # doctest: {'args': '-al', 'environ': {'DISPLAY': ':1'}}

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

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a 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
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
```

Outputs:

```
None
```

53.1.3 LibraryBaseInterface

[Link to code](#)

Inputs:

```
None
```

Outputs:

```
None
```

53.1.4 MpiCommandLine

[Link to code](#)

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'
```

Inputs:

```
[Optional]
n_procs: (an integer (int or long))
    Num processors to specify to mpiexec. Do not specify if this is
    managed externally (e.g. through SGE)
use_mpi: (a boolean, nipyne default value: False)
    Whether or not to run the command with mpiexec
environ: (a dictionary with keys which are a 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
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

None

53.1.5 SEMLikeCommandLine

[Link to code](#)

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).

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a 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
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

None

53.1.6 SimpleInterface

[Link to code](#)

An interface pattern that allows outputs to be set in a dictionary called `_results` that is automatically interpreted by `_list_outputs()` to find the outputs.

When implementing `_run_interface`, set outputs with:

```
self._results[out_name] = out_value
```

This can be a way to upgrade a Function interface to do type checking.

Examples

```
>>> from nipyne.interfaces.base import (  
...     SimpleInterface, BaseInterfaceInputSpec, TraitedSpec)
```

```
>>> def double(x):  
...     return 2 * x  
~~~  
>>> class DoubleInputSpec(BaseInterfaceInputSpec):  
...     x = traits.Float(mandatory=True)  
~~~  
>>> class DoubleOutputSpec(TraitdSpec):  
...     doubled = traits.Float()  
~~~  
>>> class Double(SimpleInterface):  
...     input_spec = DoubleInputSpec  
...     output_spec = DoubleOutputSpec  
~~~  
...     def _run_interface(self, runtime):  
...         self._results['doubled'] = double(self.inputs.x)  
...         return runtime
```

```
>>> dbl = Double()  
>>> dbl.inputs.x = 2  
>>> dbl.run().outputs.doubled  
4.0
```

Inputs:

```
None
```

Outputs:

```
None
```

53.1.7 StdOutCommandLine

[Link to code](#)

Inputs:

```
[Optional]  
out_file: (a file name)  
          argument: ``> %s``, position: -1  
args: (a unicode string)  
      Additional parameters to the command  
      argument: ``%s``  
environ: (a dictionary with keys which are a 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
```

Outputs:

None

53.2 interfaces.base.specs

53.2.1 get_filecopy_info()

[Link to code](#)

Provides information about file inputs to copy or link to cwd. Necessary for pipeline operation

53.3 interfaces.base.support

53.3.1 format_help()

[Link to code](#)

Prints help text of a Nipyype interface

```
>>> from nipyype.interfaces.afni import GCOR
>>> GCOR.help()
Wraps the executable command ``@compute_gcor``.

Computes the average correlation between every voxel
and ever other voxel, over any give mask.

For complete details, ...
```

53.3.2 get_trait_desc()

[Link to code](#)

Parses a HasTraits object into a nipyype documentation string

53.4 interfaces.base.traits_extension

53.4.1 has_metadata()

[Link to code](#)

Checks if a given trait has a metadata (and optionally if it is set to particular value)

53.4.2 isdefined()

[Link to code](#)

53.4.3 length()

[Link to code](#)

54.1 interfaces.brainsuite.brainsuite

54.1.1 BDP

[Link to code](#)

Wraps the executable 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 unicode
               string)
               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
```

(continues on next page)

(continued from previous page)

```

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
argument: ``--bvec %s --bval %s``, position: -1
mutually_exclusive: bMatrixFile
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
argument: ``%s``, position: 0
mutually_exclusive: noStructuralRegistration
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).
argument: ``--bmat %s``, position: -1
mutually_exclusive: BVecBValPair
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.
argument: ``--no-structural-registration``, position: 0
mutually_exclusive: bfcFile
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.
argument: ``--nii %s``, position: -2

[Optional]
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

```

(continues on next page)

(continued from previous page)

```

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.
argument: ``--fieldmap-correction %s``
requires: echoSpacing
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.
    argument: ``--no-distortion-correction``
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.
    argument: ``--transform-interpolation %s``
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.
    argument: ``--fieldmap-smooth3=%f``
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.
    argument: ``--transform-diffusion-surface %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.
    argument: ``--rigid-reg-measure %s``
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.
    argument: ``--output-fileprefix %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.

```

(continues on next page)

(continued from previous page)

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.

argument: ``--dcorr-regularization-wt %f``

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.

argument: ``--FRT``

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).

argument: ``--output-diffusion-coordinate``

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.

argument: ``--no-intensity-correction``

mutually_exclusive: fieldmapCorrectionMethod

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.

argument: ``--transform-t1-surface %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.

argument: ``--dir=%s``

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

(continues on next page)

(continued from previous page)

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.

argument: ``--dwi-mask %s``

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

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.
argument: ``--output-subdir %s``

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.
argument: ``--no-nonuniformity-correction``

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.
argument: ``--transform-t1-volume %s``

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.
argument: ``--low-memory``

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.
argument: ``--custom-t1-label %s``

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

(continues on next page)

(continued from previous page)

```

reports diffusivity (MD, axial, radial, L2 and L3) in a unit which
is reciprocal inverse of the unit of input b-value.
argument: ``--tensors``

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).
argument: ``--t1-mask %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.
argument: ``--bval-ratio-threshold %f``

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
argument: ``--echo-spacing=%f``

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
argument: ``--threads=%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

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.
argument: ``--flag-conf-file %s``

estimateODF_FRACT: (a boolean)

```

(continues on next page)

(continued from previous page)

```

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.
argument: ``--FRACT``

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.
argument: ``--transform-data-only``

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.
argument: ``--dcorr-reg-method %s``

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.
argument: ``--fieldmap-correction-method %s``
mutually_exclusive: skipIntensityCorr

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.
argument: ``--odf-lambda <L>``

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.
argument: ``--ignore-memory``

forcePartialROIStats: (a boolean)
The field of view (FOV) of the diffusion and T1-weighted scans may

```

(continues on next page)

(continued from previous page)

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.

argument: `--force-partial-roi-stats`

ignoreFieldmapFOV: (a boolean)

Suppresses 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.

argument: `--ignore-fieldmap-fov`

dataSinkDelay: (a list of items which are a unicode string)

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/>

argument: `--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.

argument: `--custom-label-xml %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

argument: `--transform-diffusion-volume %s`

customDiffusionLabel: (a file name)

(continues on next page)

(continued from previous page)

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`

argument: ``--custom-diffusion-label %s``

`estimateODF_3DShore`: (a float)

Estimates ODFs using 3Dshore. Pass in diffusion time, in ms

argument: ``--3dshore --diffusion_time_ms %f``

`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.

argument: ``--generate-only-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.

argument: ``--generate-stats``

Outputs:

None

54.1.2 Bfc

[Link to code](#)

Wraps the executable 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
    argument: ``-i %s``

[Optional]
biasEstimateConvergenceThreshold: (a float)
    bias estimate convergence threshold (values > 0.1 disable)
    argument: ``--beps %f``
maxBias: (a float, nipy default value: 1.5)
    maximum allowed bias value
    argument: ``-U %f``
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]
    argument: ``%s``
outputBiasField: (a file name)
    save bias field estimate
    argument: ``--bias %s``
biasFieldEstimatesOutputPrefix: (a unicode string)
    save iterative bias field estimates as <prefix>.n.field.nii.gz
    argument: ``--biasprefix %s``
inputMaskFile: (a file name)
    mask file
    argument: ``-m %s``
biasEstimateSpacing: (an integer (int or long))
    bias sample spacing (voxels)
    argument: ``-s %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
intermediate_file_type: ('analyze' or 'nifti' or 'gzippedAnalyze' or
    'gzippedNifti')
    Options for the format in which intermediate files are generated
    argument: ``%s``
minBias: (a float, nipy default value: 0.5)
    minimum allowed bias value
    argument: ``-L %f``
histogramRadius: (an integer (int or long))
```

(continues on next page)

(continued from previous page)

```

    histogram radius (voxels)
    argument: ``-r %d``
outputMRIVolume: (a file name)
    output bias-corrected MRI volume.If unspecified, output file name
    will be auto generated.
    argument: ``-o %s``
histogramType: ('ellipse' or 'block')
    Options for type of histogram
    ellipse: use ellipsoid for ROI histogram
    block :use block for ROI histogram
    argument: ``%s``
iterativeMode: (a boolean)
    iterative mode (overrides -r, -s, -c, -w settings)
    argument: ``--iterate``
convergenceThreshold: (a float)
    convergence threshold
    argument: ``--eps %f``
correctedImagesOutputPrefix: (a unicode string)
    save iterative corrected images as <prefix>.n.bfc.nii.gz
    argument: ``--prefix %s``
correctionScheduleFile: (a file name)
    list of parameters
    argument: ``--schedule %s``
splineLambda: (a float)
    spline stiffness weighting parameter
    argument: ``-w %f``
controlPointSpacing: (an integer (int or long))
    control point spacing (voxels)
    argument: ``-c %d``
outputMaskedBiasField: (a file name)
    save bias field estimate (masked)
    argument: ``--maskedbias %s``
correctWholeVolume: (a boolean)
    apply correction field to entire volume
    argument: ``--extrapolate``
timer: (a boolean)
    display timing information
    argument: ``--timer``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbosityLevel: (an integer (int or long))
    verbosity level (0=silent)
    argument: ``-v %d``

```

Outputs:

```

outputMRIVolume: (a file name)
    path/name of output file
outputBiasField: (a file name)
    path/name of bias field output file
outputMaskedBiasField: (a file name)
    path/name of masked bias field output
correctionScheduleFile: (a file name)
    path/name of schedule file

```

54.1.3 Bse

[Link to code](#)

Wraps the executable 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
    argument: ``-i %s``

[Optional]
noRotate: (a boolean)
    retain original orientation(default behavior will auto-rotate input
    NII files to LPI orientation)
    argument: ``--norotate``
outputDetailedBrainMask: (a file name)
    save detailed brain mask
    argument: ``--hires %s``
trim: (a boolean, nipy default value: True)
    trim brainstem
    argument: ``--trim``
timer: (a boolean)
    show timing
    argument: ``--timer``
radius: (a float, nipy default value: 1)
    radius of erosion/dilation filter
    argument: ``-r %f``
outputDiffusionFilter: (a file name)
    diffusion filter output
    argument: ``--adf %s``
outputEdgeMap: (a file name)
    edge map output
    argument: ``--edge %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputCortexFile: (a file name)
    cortex file
    argument: ``--cortex %s``
outputMaskFile: (a file name)
    save smooth brain mask. If unspecified, output file name will be
    auto generated.
    argument: ``--mask %s``
verbosityLevel: (a float, nipy default value: 1)
```

(continues on next page)

(continued from previous page)

```

        verbosity level (0=silent)
        argument: ``-v %f``
outputMRIVolume: (a file name)
        output brain-masked MRI volume. If unspecified, output file name
        will be auto generated.
        argument: ``-o %s``
diffusionIterations: (an integer (int or long), nipy default value:
        3)
        diffusion iterations
        argument: ``-n %d``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
edgeDetectionConstant: (a float, nipy default value: 0.64)
        edge detection constant
        argument: ``-s %f``
diffusionConstant: (a float, nipy default value: 25)
        diffusion constant
        argument: ``-d %f``
dilateFinalMask: (a boolean, nipy default value: True)
        dilate final mask
        argument: ``-p``

```

Outputs:

```

outputCortexFile: (a file name)
        path/name of cortex file
outputDetailedBrainMask: (a file name)
        path/name of detailed brain mask
outputMaskFile: (a file name)
        path/name of smooth brain mask
outputMRIVolume: (a file name)
        path/name of brain-masked MRI volume
outputDiffusionFilter: (a file name)
        path/name of diffusion filter output
outputEdgeMap: (a file name)
        path/name of edge map output

```

54.1.4 Cerebro[Link to code](#)

Wraps the executable 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]
inputAtlasMRIFile: (a file name)
    atlas MRI volume
    argument: ``--atlas %s``
inputAtlasLabelFile: (a file name)
    atlas labeling
    argument: ``--atlaslabels %s``
inputMRIFile: (a file name)
    input 3D MRI volume
    argument: ``-i %s``

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outputAffineTransformFile: (a file name)
    save affine transform to file.
    argument: ``--air %s``
outputWarpTransformFile: (a file name)
    save warp transform to file.
    argument: ``--warp %s``
warpLabel: (an integer (int or long))
    warp order (2,3,4,5,6,7,8)
    argument: ``--warplevel %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
outputCerebrumMaskFile: (a file name)
    output cerebrum mask volume. If unspecified, output file name will
    be auto generated.
    argument: ``-o %s``
linearConvergence: (a float)
    linear convergence
    argument: ``--linconv %f``
warpConvergence: (a float)
    warp convergence
    argument: ``--warpconv %f``
outputLabelVolumeFile: (a file name)
    output labeled hemisphere/cerebrum volume. If unspecified, output
    file name will be auto generated.
    argument: ``-l %s``
useCentroids: (a boolean)
    use centroids of data to initialize position
    argument: ``--centroids``
inputBrainMaskFile: (a file name)
    brain mask file
    argument: ``-m %s``
tempDirectoryBase: (a unicode string)
    create a temporary directory within this directory
    argument: ``--tempdirbase %s``
verbosity: (an integer (int or long))
    verbosity level (0=silent)
    argument: ``-v %d``
costFunction: (an integer (int or long), nipy default value: 2)
    0,1,2
```

(continues on next page)

(continued from previous page)

```

        argument: ``-c %d``
keepTempFiles: (a boolean)
    don't remove temporary files
    argument: ``--keep``
tempDirectory: (a unicode string)
    specify directory to use for temporary files
    argument: ``--tempdir %s``

```

Outputs:

```

outputWarpTransformFile: (a file name)
    path/name of warp transform file
outputLabelVolumeFile: (a file name)
    path/name of label mask file
outputCerebrumMaskFile: (a file name)
    path/name of cerebrum mask file
outputAffineTransformFile: (a file name)
    path/name of affine transform file

```

54.1.5 Cortex[Link to code](#)Wraps the executable 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]
inputTissueFractionFile: (a file name)
    tissue fraction file (32-bit float)
    argument: ``-f %s``
inputHemisphereLabelFile: (a file name)
    hemisphere / lobe label volume
    argument: ``-h %s``

[Optional]
includeAllSubcorticalAreas: (a boolean, nipy default value: True)
    include all subcortical areas in WM mask
    argument: ``-a``
computeGCBoundary: (a boolean)
    compute GM/CSF boundary
    argument: ``-g``
timer: (a boolean)
    timing function
    argument: ``--timer``

```

(continues on next page)

(continued from previous page)

```

verbosity: (an integer (int or long))
    verbosity level
    argument: ``-v %d``
tissueFractionThreshold: (a float, nipy default value: 50.0)
    tissue fraction threshold (percentage)
    argument: ``-p %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
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
computeWGBoundary: (a boolean, nipy default value: True)
    compute WM/GM boundary
    argument: ``-w``
outputCerebrumMask: (a file name)
    output structure mask. If unspecified, output file name will be auto
    generated.
    argument: ``-o %s``

```

Outputs:

```

outputCerebrumMask: (a file name)
    path/name of cerebrum mask

```

54.1.6 Dewisp[Link to code](#)

Wraps the executable command dewisp.

dewisp removes wispy tendrill structures 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
    argument: ``-i %s``

[Optional]
maximumIterations: (an integer (int or long))
    maximum number of iterations
    argument: ``-n %d``
sizeThreshold: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

        size threshold
        argument: ``-t %d``
timer: (a boolean)
        time processing
        argument: ``--timer``
outputMaskFile: (a file name)
        output file. If unspecified, output file name will be auto
        generated.
        argument: ``-o %s``
verbosity: (an integer (int or long))
        verbosity
        argument: ``-v %d``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

outputMaskFile: (a file name)
        path/name of mask file

```

54.1.7 Dfs[Link to code](#)Wraps the executable 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
        argument: ``-i %s``

[Optional]
postSmoothFlag: (a boolean)
        smooth vertices after coloring
        argument: ``--postsmooth``
noNormalsFlag: (a boolean)
        do not compute vertex normals
        argument: ``--nonormals``
specialTessellation: ('greater_than' or 'less_than' or 'equal_to')
        To avoid throwing a UserWarning, set tessellationThreshold first.

```

(continues on next page)

(continued from previous page)

```

    Then set this attribute.
    Usage: tessellate voxels greater_than, less_than, or equal_to
    <tessellationThreshold>
    argument: ``%s``, position: -1
    mutually_exclusive: nonZeroTessellation, specialTessellation
    requires: tessellationThreshold
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
nonZeroTessellation: (a boolean)
    tessellate non-zero voxels
    argument: ``-nz``
    mutually_exclusive: nonZeroTessellation, specialTessellation
curvatureWeighting: (a float, nipy default value: 5.0)
    curvature weighting
    argument: ``-w %f``
smoothingIterations: (an integer (int or long), nipy default value:
    10)
    number of smoothing iterations
    argument: ``-n %d``
inputShadingVolume: (a file name)
    shade surface model with data from image volume
    argument: ``-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
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>
    argument: ``%f``
zeroPadFlag: (a boolean)
    zero-pad volume (avoids clipping at edges)
    argument: ``-z``
scalingPercentile: (a float)
    scaling percentile
    argument: ``-f %f``
timer: (a boolean)
    timing function
    argument: ``--timer``
outputSurfaceFile: (a file name)
    output surface mesh file. If unspecified, output file name will be
    auto generated.
    argument: ``-o %s``
verbosity: (an integer (int or long))
    verbosity (0 = quiet)
    argument: ``-v %d``
smoothingConstant: (a float, nipy default value: 0.5)
    smoothing constant
    argument: ``-a %f``

```

Outputs:

```

outputSurfaceFile: (a file name)
    path/name of surface file

```

54.1.8 Hemisplit

[Link to code](#)

Wraps the executable 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]
inputSurfaceFile: (a file name)
    input surface
    argument: ``-i %s``
inputHemisphereLabelFile: (a file name)
    input hemisphere label volume
    argument: ``-l %s``

[Optional]
outputRightHemisphere: (a file name)
    output surface file, right hemisphere. If unspecified, output file
    name will be auto generated.
    argument: ``--right %s``
outputLeftHemisphere: (a file name)
    output surface file, left hemisphere. If unspecified, output file
    name will be auto generated.
    argument: ``--left %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
timer: (a boolean)
    timing function
    argument: ``--timer``
verbosity: (an integer (int or long))
    verbosity (0 = silent)
    argument: ``-v %d``
outputRightPialHemisphere: (a file name)
    output pial surface file, right hemisphere. If unspecified, output
    file name will be auto generated.
    argument: ``-pr %s``
pialSurfaceFile: (a file name)
    pial surface file -- must have same geometry as input surface
    argument: ``-p %s``
outputLeftPialHemisphere: (a file name)
    output pial surface file, left hemisphere. If unspecified, output
    file name will be auto generated.
    argument: ``-pl %s``
environ: (a dictionary with keys which are a bytes or None or a value
```

(continues on next page)

(continued from previous page)

```

of class 'str' and with values which are a bytes or None or a
value of class 'str', nipy default value: {})
Environment variables

```

Outputs:

```

outputRightPialHemisphere: (a file name)
    path/name of right pial hemisphere
outputRightHemisphere: (a file name)
    path/name of right hemisphere
outputLeftHemisphere: (a file name)
    path/name of left hemisphere
outputLeftPialHemisphere: (a file name)
    path/name of left pial hemisphere

```

54.1.9 Pialmesh[Link to code](#)

Wraps the executable 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 nipy.interfaces import brainsuite
>>> from nipy.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]
inputTissueFractionFile: (a file name)
    floating point (32) tissue fraction image
    argument: ``-f %s``
inputSurfaceFile: (a file name)
    input file
    argument: ``-i %s``
inputMaskFile: (a file name)
    restrict growth to mask file region
    argument: ``-m %s``

[Optional]
tangentSmoother: (a float)
    strength of tangential smoother.
    argument: ``--tc %f``
searchRadius: (a float, nipy default value: 1)
    search radius
    argument: ``-r %f``
recomputeNormals: (a boolean)
    recompute normals at each iteration
    argument: ``--norm``
timer: (a boolean)

```

(continues on next page)

(continued from previous page)

```

    show timing
    argument: ``--timer``
outputSurfaceFile: (a file name)
    output file. If unspecified, output file name will be auto
    generated.
    argument: ``-o %s``
tissueThreshold: (a float, nipype default value: 1.05)
    tissue threshold
    argument: ``-t %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', nipype default value: {})
    Environment variables
outputInterval: (an integer (int or long), nipype default value: 10)
    output interval
    argument: ``--interval %d``
stepSize: (a float, nipype default value: 0.4)
    step size
    argument: ``-s %f``
maxThickness: (a float, nipype default value: 20)
    maximum allowed tissue thickness
    argument: ``--max %f``
exportPrefix: (a unicode string)
    prefix for exporting surfaces if interval is set
    argument: ``--prefix %s``
laplacianSmoothing: (a float, nipype default value: 0.025)
    apply Laplacian smoothing
    argument: ``--smooth %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbosity: (an integer (int or long))
    verbosity
    argument: ``-v %d``
numIterations: (an integer (int or long), nipype default value: 100)
    number of iterations
    argument: ``-n %d``
normalSmoother: (a float, nipype default value: 0.2)
    strength of normal smoother.
    argument: ``--nc %f``

```

Outputs:

```

outputSurfaceFile: (a file name)
    path/name of surface file

```

54.1.10 Pvc[Link to code](#)

Wraps the executable 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
    argument: ``-i %s``

[Optional]
outputTissueFractionFile: (a file name)
    output tissue fraction file
    argument: ``-f %s``
threeClassFlag: (a boolean)
    use a three-class (CSF=0,GM=1,WM=2) labeling
    argument: ``-3``
verbosity: (an integer (int or long))
    verbosity level (0 = silent)
    argument: ``-v %d``
spatialPrior: (a float)
    spatial prior strength
    argument: ``-l %f``
timer: (a boolean)
    time processing
    argument: ``--timer``
outputLabelFile: (a file name)
    output label file. If unspecified, output file name will be auto
    generated.
    argument: ``-o %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputMaskFile: (a file name)
    brain mask file
    argument: ``-m %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
outputLabelFile: (a file name)
    path/name of label file
outputTissueFractionFile: (a file name)
    path/name of tissue fraction file
```

54.1.11 SVReg

[Link to code](#)

Wraps the executable 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
    argument: ``'%s'``, position: 0

[Optional]
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
    argument: ``'%s'``, position: 1
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.
    argument: ``'-cbm'``
shortMessages: (a boolean)
    Short messages instead of detailed messages
    argument: ``'-gui'``
keepIntermediates: (a boolean)
    Keep the intermediate files after the svreg sequence is complete.
    argument: ``'-k'``
verbosity0: (a boolean)
    no messages will be reported
    argument: ``'-v0'``
    mutually_exclusive: verbosity0, verbosity1, verbosity2
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
verbosity2: (a boolean)
    all the messages, including per-iteration, will be displayed
    argument: ``'v2'``
    mutually_exclusive: verbosity0, verbosity1, verbosity2
skipToVolumeReg: (a boolean)
    If surface registration was already performed at an earlier time and
```

(continues on next page)

(continued from previous page)

```

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.
argument: ``'-s'``
displayTimestamps: (a boolean)
Timestamps will be displayed in the messages
argument: ``'-t'``
skipVolumetricProcessing: (a boolean)
Only surface registration and labeling will be performed. Volumetric
processing will be skipped.
argument: ``'-S'``
verbosity1: (a boolean)
messages will be reported but not the iteration-wise detailed
messages
argument: ``'-v1'``
mutually_exclusive: verbosity0, verbosity1, verbosity2
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.
argument: ``'-C'``
useMultiThreading: (a boolean)
If multiple CPUs are present on the system, the code will try to use
multithreading to make the execution fast.
argument: ``'-p'``
useSingleThreading: (a boolean)
Use single threaded mode.
argument: ``'-U'``
dataSinkDelay: (a list of items which are a unicode string)
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
argument: ``'%s'``
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"
argument: ``'-cur %s'``
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)
argument: ``'-D %d'``
displayModuleName: (a boolean)
Module name will be displayed in the messages
argument: ``'-m'``
args: (a unicode string)
Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
refineOutputs: (a boolean)
    Refine outputs at the expense of more processing time.
    argument: ``'-r'``
iterations: (an integer (int or long))
    Assigns a number of iterations in the intensity registration step.if
    unspecified, performs 100 iterations
    argument: ``'-H %d'``
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.
    argument: ``'-p'``

```

Outputs:**None**

54.1.12 Scrubmask

[Link to code](#)

Wraps the executable 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
    argument: ``-i %s``

[Optional]
numberIterations: (an integer (int or long))
    number of iterations
    argument: ``-n %d``
outputMaskFile: (a file name)
    output structure mask file. If unspecified, output file name will be
    auto generated.
    argument: ``-o %s``
timer: (a boolean)
    timing function
    argument: ``--timer``
verbosity: (an integer (int or long))
    verbosity (0=silent)
    argument: ``-v %d``

```

(continues on next page)

(continued from previous page)

```
backgroundFillThreshold: (an integer (int or long), nipy default
    value: 2)
    background fill threshold
    argument: ``-b %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
foregroundTrimThreshold: (an integer (int or long), nipy default
    value: 0)
    foreground trim threshold
    argument: ``-f %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
```

Outputs:

```
outputMaskFile: (a file name)
    path/name of mask file
```

54.1.13 Skullfinder[Link to code](#)

Wraps the executable 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]
inputMaskFile: (a file name)
    A brain mask file, 8-bit image (0=non-brain, 255=brain)
    argument: ``-m %s``
inputMRIFile: (a file name)
    input file
    argument: ``-i %s``

[Optional]
lowerThreshold: (an integer (int or long))
    Lower threshold for segmentation
    argument: ``-l %d``
scalpLabelValue: (an integer (int or long))
    scalp label value (0-255)
    argument: ``--scalplabel %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

(continues on next page)

(continued from previous page)

```

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.
    argument: ``-o %s``
surfaceFilePrefix: (a unicode string)
    if specified, generate surface files for brain, skull, and scalp
    argument: ``-s %s``
brainLabelValue: (an integer (int or long))
    brain label value (0-255)
    argument: ``--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', nipype default value: {})
    Environment variables
performFinalOpening: (a boolean)
    perform a final opening operation on the scalp mask
    argument: ``--finalOpening``
spaceLabelValue: (an integer (int or long))
    space label value (0-255)
    argument: ``--spacelabel %d``
bgLabelValue: (an integer (int or long))
    background label value (0-255)
    argument: ``--bglabel %d``
verbosity: (an integer (int or long))
    verbosity
    argument: ``-v %d``
upperThreshold: (an integer (int or long))
    Upper threshold for segmentation
    argument: ``-u %d``
skullLabelValue: (an integer (int or long))
    skull label value (0-255)
    argument: ``--skulllabel %d``

```

Outputs:

```

outputLabelFile: (a file name)
    path/name of label file

```

54.1.14 Tca[Link to code](#)Wraps the executable command `tca`.

topological correction algorithm (TCA) This program removes topological handles from a binary object.

<http://brainsuite.org/processing/surfaceextraction/tca/>**Examples**

```

>>> from nipype.interfaces import brainsuite
>>> from nipype.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
    argument: ``-i %s``

[Optional]
timer: (a boolean)
    timing function
    argument: ``--timer``
outputMaskFile: (a file name)
    output mask volume. If unspecified, output file name will be auto
    generated.
    argument: ``-o %s``
verbosity: (an integer (int or long))
    verbosity (0 = quiet)
    argument: ``-v %d``
maxCorrectionSize: (an integer (int or long))
    minimum correction size
    argument: ``-n %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
foregroundDelta: (an integer (int or long), nipy default value: 20)
    foreground delta
    argument: ``--delta %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
minCorrectionSize: (an integer (int or long), nipy default value:
    2500)
    maximum correction size
    argument: ``-m %d``

```

Outputs:

```

outputMaskFile: (a file name)
    path/name of mask file

```

54.1.15 ThicknessPVC**Link to code**

Wraps the executable 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
    argument: ``%s``

[Optional]
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

None

54.1.16 getFileName()

[Link to code](#)

54.1.17 l_outputs()

[Link to code](#)

55.1 interfaces.camino.calib

55.1.1 SFLUTGen

[Link to code](#)

Wraps the executable 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 nipyype.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]
info_file: (a file name)
    The Info file that corresponds to the calibration datafile used in
    the reconstruction.
    argument: ``-infofile %s``
in_file: (an existing file name)
    Voxel-order data of the spherical functions peaks.
    argument: ``-inputfile %s``

[Optional]
outputstem: (a unicode string, nipyype default value: LUT)
    Define the name of the generated luts. The form of the filenames
    will be [outputstem]_oneFibreSurfaceCoeffs.Bdouble and
    [outputstem]_twoFibreSurfaceCoeffs.Bdouble
    argument: ``-outputstem %s``
out_file: (a file name)
    argument: ``> %s``, position: -1
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.
    argument: ``-minvectsperbin %d``
pdf: ('bingham' or 'watson', nipyype 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.
    argument: ``-pdf %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
order: (an integer (int or long))
    The order of the polynomial fitting the surface. Order 1 is linear.
    Order 2 (default) is quadratic.
    argument: ``-order %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
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.
```

(continues on next page)

(continued from previous page)

```

        argument: ``-binincsize %d``
directmap: (a boolean)
        Use direct mapping between the eigenvalues and the distribution
        parameters instead of the log of the eigenvalues.
        argument: ``-directmap``

```

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

```

55.1.2 SFPICOCalibData[Link to code](#)Wraps the executable 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.
    argument: ``-infooutputfile %s``
scheme_file: (an existing file name)
    Specifies the scheme file for the diffusion MRI data
    argument: ``-schemefile %s``

[Optional]
twodtfstep: (a float)
    FA step size controlling how many steps there are between the
    minimum and maximum FA settings for the two tensor cases.
    argument: ``-twodtfstep %f``
out_file: (a file name)
    argument: ``> %s``, position: -1
onedtfstep: (a float)
    FA step size controlling how many steps there are between the
    minimum and maximum FA settings.
    argument: ``-onedtfstep %f``
twodtanglerange: (a list of from 2 to 2 items which are a float)
    Minimum and maximum crossing angles between the two fibres.
    argument: ``-twodtanglerange %s``
twodtmixmax: (a float)
    Mixing parameter controlling the proportion of one fibre population
    to the other. The minimum mixing parameter is (1 - twodtmixmax).
    argument: ``-twodtmixmax %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
twodtmixstep: (a float)
    Mixing parameter step size for the two tensor cases. Specify how
    many mixing parameter increments to use.
    argument: ``-twodtmixstep %f``
trace: (a float)
    Trace of the diffusion tensor(s) used in the test function.
    argument: ``-trace %f``
seed: (a float)
    Specifies the random seed to use for noise generation in simulation
    trials.
    argument: ``-seed %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.
    argument: ``-twodtfarange %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``-twodtanglestep %f``
snr: (a float)
```

(continues on next page)

(continued from previous page)

```

    Specifies the signal-to-noise ratio of the non-diffusion-weighted
    measurements to use in simulations.
    argument: ``-snr %f``
onedtfarange: (a list of from 2 to 2 items which are a float)
    Minimum and maximum FA for the single tensor synthetic data.
    argument: ``-onedtfarange %s``

```

Outputs:

```

calib_info: (an existing file name)
    Calibration dataset
PICOcalib: (an existing file name)
    Calibration dataset

```

55.2 interfaces.camino.connectivity

55.2.1 Conmat

[Link to code](#)

Wraps the executable 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
    argument: ``-inputfile %s``
target_file: (an existing file name)
    An image containing targets, as used in ProcStreamlines interface.
    argument: ``-targetfile %s``

[Optional]
tract_stat: ('mean' or 'min' or 'max' or 'sum' or 'median' or 'var')
    Tract statistic to use. See TractStats for other options.
    argument: ``-tractstat %s``
    mutually_exclusive: tract_prop
    requires: scalar_file
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.
    argument: ``-targetnamefile %s``
tract_prop: ('length' or 'endpointsep')
    Tract property average to compute in the connectivity matrix. See
    TractStats for details.
    argument: ``-tractstat %s``
    mutually_exclusive: tract_stat
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
```

(continues on next page)

(continued from previous page)

```

        value of class 'str', nipy default value: {})
    Environment variables
scalar_file: (an existing file name)
    Optional scalar file for computing tract-based statistics. Must be
    in the same space as the target file.
    argument: ``-scalarfile %s``
    requires: tract_stat
output_root: (a file name)
    filename root prepended onto the names of the output files. The
    extension will be determined from the input.
    argument: ``-outputroot %s``

```

Outputs:

```

conmat_sc: (an existing file name)
    Connectivity matrix in CSV file.
conmat_ts: (a file name)
    Tract statistics in CSV file.

```

55.3 interfaces.camino.convert

55.3.1 AnalyzeHeader

[Link to code](#)

Wraps the executable 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 nipy.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.
    argument: ``-datatype %s``
in_file: (an existing file name)
    Tensor-fitted data filename

```

(continues on next page)

(continued from previous page)

```

        argument: ``< %s``, position: 1

[Optional]
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].
    argument: ``-picoseed %s``
nimages: (an integer (int or long))
    Number of images in the img file. Default 1.
    argument: ``-nimages %d``
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.
    argument: ``-printprogargs %s``, 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
networkbyteorder: (a boolean)
    Write header in network byte order (big-endian). This is the default
    for new headers.
    argument: ``-networkbyteorder``
printbigendian: (an existing file name)
    Prints 1 if the header is big-endian, 0 otherwise.
    argument: ``-printbigendian %s``, position: 3
printintelbyteorder: (an existing file name)
    Prints 1 if the header is little-endian, 0 otherwise.
    argument: ``-printintelbyteorder %s``, position: 3
data_dims: (a list of from 3 to 3 items which are an integer (int or
long))
    data dimensions in voxels
    argument: ``-datadims %s``
scheme_file: (an existing file name)
    Camino scheme file (b values / vectors, see camino.fsl2scheme)
    argument: ``%s``, position: 2
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    argument: ``-voxeldims %s``
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.
    argument: ``-gl %s``
description: (a string)
    Short description - No spaces, max length 79 bytes. Will be null
    terminated automatically.
    argument: ``-description %s``
out_file: (a file name)
    argument: ``> %s``, position: -1
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.
    argument: ``-initfromheader %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

```

(continues on next page)

(continued from previous page)

```

the other arguments.
argument: ``-readheader %s``, position: 3
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].
        argument: ``-centre %s``
scaleinter: (a float)
        Constant to add to the image intensities. Used by SPM and MRICro.
        argument: ``-scaleinter %d``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
printimagedims: (an existing file name)
        Prints image data and voxel dimensions as Camino arguments and
        exits.
        argument: ``-printimagedims %s``, position: 3
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.
        argument: ``-offset %d``
intelbyteorder: (a boolean)
        Write header in intel byte order (little-endian).
        argument: ``-intelbyteorder``
scaleslope: (a float)
        Intensities in the image are scaled by this factor by SPM and
        MRICro. Default is 1.0.
        argument: ``-scaleslope %d``

```

Outputs:

```

header: (an existing file name)
        Analyze header

```

55.3.2 DT2NIFTI[Link to code](#)Wraps the executable 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
        argument: ``-header %s``, position: 3
in_file: (an existing file name)
        tract file
        argument: ``-inputfile %s``, position: 1

[Optional]
output_root: (a file name)
        filename root prepended onto the names of three output files.
        argument: ``-outputroot %s``, position: 2

```

(continues on next page)

(continued from previous page)

```
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
lms0: (an existing file name)
    estimated lms0 from Camino reconstruction in NIfTI format
dt: (an existing file name)
    diffusion tensors in NIfTI format
exitcode: (an existing file name)
    exit codes from Camino reconstruction in NIfTI format
```

55.3.3 Image2Voxel[Link to code](#)

Wraps the executable 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
    argument: ``-4dimage %s``, position: 1

[Optional]
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"
    argument: ``-outputdatatype %s``, position: 2
out_file: (a file name)
    argument: ``> %s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
voxel_order: (an existing file name)
              path/name of 4D volume in voxel order
```

55.3.4 NiftIDT2Camino

[Link to code](#)

Wraps the executable command `niftidt2camino`.

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, dyy, 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.
        argument: ``-inputfile %s``, position: 1

[Optional]
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.
        argument: ``-s0 %s``
out_file: (a file name)
        argument: ``> %s``, position: -1
uppertriangular: (a boolean)
        Specifies input in upper-triangular (VTK style) order.
        argument: ``-uppertriangular %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%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.
        argument: ``-scaleslope %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.
        argument: ``-scaleinter %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.
        argument: ``-bgmask %s``
lns0_file: (an existing file name)
        File containing the log of the unweighted signal for each voxel, may
```

(continues on next page)

(continued from previous page)

```

be a raw binary file (specify type with -inputdatatype) or a
supported image file.
argument: ``-lns0 %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
Environment variables

```

Outputs:

```

out_file: (a file name)
          diffusion tensors data in Camino format

```

55.3.5 ProcStreamlines

[Link to code](#)Wraps the executable 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
         argument: ``-inputfile %s``, position: 1

[Optional]
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
              argument: ``-seedpointmm %s``
seedfile: (a file name)
           Image Containing Seed Points
           argument: ``-seedfile %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.
               argument: ``-waypointfile %s``
inputmodel: ('raw' or 'voxels', nipy default value: raw)
             input model type (raw or voxels)
             argument: ``-inputmodel %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

```

(continues on next page)

(continued from previous page)

```

    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).
    argument: ``-directional %s``
voxeldims: (a list of from 3 to 3 items which are an integer (int or
    long))
    voxel dimensions in mm
    argument: ``-voxeldims %s``
outputroot: (a file name)
    Prepended onto all output file names.
    argument: ``-outputroot %s``
gzip: (a boolean)
    save the output image in gzip format
    argument: ``-gzip``
outputsc: (a boolean)
    output the connection probability map (raw streamlines, int)
    argument: ``-outputsc``
    requires: outputroot, seedfile
outputacm: (a boolean)
    output all tracts in a single connection probability map (Analyze
    image)
    argument: ``-outputacm``
    requires: outputroot, seedfile
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.
    argument: ``-discardloops``
outputtcp: (a boolean)
    output the connection probability map (Analyze image, float)
    argument: ``-outputtcp``
    requires: outputroot, seedfile
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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.
    argument: ``-resamplestepsize %d``
allowmultitargets: (a boolean)
    Allows streamlines to connect to multiple target volumes.
    argument: ``-allowmultitargets``
targetfile: (a file name)
    Image containing target volumes.
    argument: ``-targetfile %s``
maxtractlength: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

        maximum length of tracts
        argument: ``-maxtractlength %d``
mintractpoints: (an integer (int or long))
        minimum number of tract points
        argument: ``-mintractpoints %d``
outputtracts: (a boolean)
        Output streamlines in raw binary format.
        argument: ``-outputtracts``
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.
        argument: ``-truncateloops``
maxtractpoints: (an integer (int or long))
        maximum number of tract points
        argument: ``-maxtractpoints %d``
outputcbs: (a boolean)
        outputs connectivity-based segmentation maps; requires target
        outputfile
        argument: ``-outputcbs``
        requires: outputroot, targetfile, seedfile
exclusionfile: (a file name)
        Image containing exclusion ROIs. This should be an Analyze 7.5
        header / image file.hdr and file.img.
        argument: ``-exclusionfile %s``
endpointfile: (a file name)
        Image containing endpoint ROIs. This should be an Analyze 7.5 header
        / image file.hdr and file.img.
        argument: ``-endpointfile %s``
mintractlength: (an integer (int or long))
        minimum length of tracts
        argument: ``-mintractlength %d``
regionindex: (an integer (int or long))
        index of specific region to process
        argument: ``-regionindex %d``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
datadims: (a list of from 3 to 3 items which are an integer (int or
        long))
        data dimensions in voxels
        argument: ``-datadims %s``
truncateinexclusion: (a boolean)
        Retain segments of a streamline before entry to an exclusion ROI.
        argument: ``-truncateinexclusion``
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
        argument: ``-seedpointvox %s``
out_file: (a file name)
        argument: ``> %s``, position: -1
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.
        argument: ``-iterations %d``

```

(continues on next page)

(continued from previous page)

```
noresample: (a boolean)
    Disables resampling of input streamlines. Resampling is
    automatically disabled if the input model is voxels.
    argument: ``-noresample``
```

Outputs:

```
proc: (an existing file name)
    Processed Streamlines
outputroot_files: (a list of items which are an existing file name)
```

55.3.6 Shredder[Link to code](#)

Wraps the executable 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 nipy.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
    argument: ``< %s``, position: -2

[Optional]
space: (an integer (int or long))
    skips space bytes
    argument: ``%d``, position: 3
out_file: (a file name)
    argument: ``> %s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
offset: (an integer (int or long))
    initial offset of offset bytes
    argument: ``%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
```

(continues on next page)

(continued from previous page)

```

chunksize: (an integer (int or long))
            reads and outputs a chunk of chunksize bytes
            argument: ``%d``, position: 2

```

Outputs:

```

shredded: (an existing file name)
           Shredded binary data file

```

55.3.7 TractShredder[Link to code](#)

Wraps the executable 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
         argument: ``< %s``, position: -2

[Optional]
space: (an integer (int or long))
       skips space tracts
       argument: ``%d``, position: 3
offset: (an integer (int or long))
        initial offset of offset tracts
        argument: ``%d``, position: 1
out_file: (a file name)
          argument: ``> %s``, position: -1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
bunchsize: (an integer (int or long))
           reads and outputs a group of bunchsize tracts
           argument: ``%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

```

Outputs:

```
shredded: (an existing file name)
          Shredded tract file
```

55.3.8 VtkStreamlines

[Link to code](#)

Wraps the executable command `vtkstreamlines`.

Use `vtkstreamlines` to convert raw or voxel format streamlines to VTK polydata

Examples

```
>>> import nipyre.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
         argument: `` < %s``, position: -2

[Optional]
voxeldims: (a list of from 3 to 3 items which are an integer (int or
           long))
           voxel dimensions in mm
           argument: ``-voxeldims %s``, position: 4
colourorient: (a boolean)
              Each point on the streamline is coloured by the local orientation.
              argument: ``-colourorient``
out_file: (a file name)
          argument: ``> %s``, position: -1
seed_file: (a file name)
           image containing seed points
           argument: ``-seedfile %s``, position: 1
inputmodel: ('raw' or 'voxels', nipyre default value: raw)
            input model type (raw or voxels)
            argument: ``-inputmodel %s``
environ: (a dictionary with keys which are a 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
interpolate: (a boolean)
            the scalar value at each point on the streamline is calculated by
            trilinear interpolation
            argument: ``-interpolate``
target_file: (a file name)
            image containing integer-valued target regions
            argument: ``-targetfile %s``, position: 2
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
interpolatescalars: (a boolean)
                   the scalar value at each point on the streamline is calculated by
```

(continues on next page)

(continued from previous page)

```

    trilinear interpolation
    argument: ``-interpolatescalars``
scalar_file: (a file name)
    image that is in the same physical space as the tracts
    argument: ``-scalarfile %s``, position: 3

```

Outputs:

```

vtk: (an existing file name)
    Streamlines in VTK format

```

55.4 interfaces.camino.dti

55.4.1 ComputeEigensystem

[Link to code](#)

Wraps the executable 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: {l₁, e₁₁, e₁₂, e₁₃, l₂, e₂₁, e₂₂, e₂₃, l₃, e₃₁, e₃₂, e₃₃}, where l₁ ≥ l₂ ≥ l₃ 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
    argument: ``< %s``, position: 1

[Optional]
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
    argument: ``-inputdatatype %s``
maxcomponents: (an integer (int or long))
    The maximum number of tensor components in a voxel of the input
    data.
    argument: ``-maxcomponents %d``
out_file: (a file name)
    argument: ``> %s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

```

(continues on next page)

(continued from previous page)

```
inputmodel: ('dt' or 'multitensor')
    Specifies the model that the input data contains parameters for.
    Possible model types are: "dt" (diffusion-tensor data) and
    "multitensor"
    argument: ``-inputmodel %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
    argument: ``-outputdatatype %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
eigen: (an existing file name)
    Trace of the diffusion tensor
```

55.4.2 ComputeFractionalAnisotropy[Link to code](#)Wraps the executable 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
    argument: ``< %s``, position: 1

[Optional]
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".
    argument: ``-inputdatatype %s``
out_file: (a file name)
    argument: ``> %s``, position: -1
args: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

Additional parameters to the command
argument: ``%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.
    argument: ``-inputmodel %s``
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".
    argument: ``-outputdatatype %s``
scheme_file: (an existing file name)
    Camino scheme file (b values / vectors, see camino.fsl2scheme)
    argument: ``%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

```

Outputs:

```

fa: (an existing file name)
    Fractional Anisotropy Map

```

55.4.3 ComputeMeanDiffusivity[Link to code](#)Wraps the executable 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
    argument: ``< %s``, position: 1

[Optional]
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".
    argument: ``-inputdatatype %s``

```

(continues on next page)

(continued from previous page)

```

out_file: (a file name)
    argument: ``> %s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``-inputmodel %s``
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".
    argument: ``-outputdatatype %s``
scheme_file: (an existing file name)
    Camino scheme file (b values / vectors, see camino.fsl2scheme)
    argument: ``%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

```

Outputs:

```

md: (an existing file name)
    Mean Diffusivity Map

```

55.4.4 ComputeTensorTrace[Link to code](#)Wraps the executable 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

```

(continues on next page)

(continued from previous page)

```

    argument: ``< %s``, position: 1

[Optional]
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".
    argument: ``-inputdatatype %s``
out_file: (a file name)
    argument: ``> %s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``-inputmodel %s``
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".
    argument: ``-outputdatatype %s``
scheme_file: (an existing file name)
    Camino scheme file (b values / vectors, see camino.fsl2scheme)
    argument: ``%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

```

Outputs:

```

trace: (an existing file name)
    Trace of the diffusion tensor

```

55.4.5 DTIFit[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: 1
scheme_file: (an existing file name)
             Camino scheme file (b values / vectors, see camino.fsl2scheme)
             argument: ``%s``, position: 2

[Optional]
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.
        argument: ``-bgmask %s``
out_file: (a file name)
          argument: ``> %s``, position: -1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
non_linear: (a boolean)
           Use non-linear fitting instead of the default linear regression to
           the log measurements.
           argument: ``-nonlinear``, position: 3
```

Outputs:

```
tensor_fitted: (an existing file name)
              path/name of 4D volume in voxel order
```

55.4.6 DTLUTGen

[Link to code](#)

Wraps the executable 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.
    argument: ``-schemefile %s``, position: 2

[Optional]
samples: (an integer (int or long))
    The number of synthetic measurements to generate at each point in
    the LUT. The default is 2000.
    argument: ``-samples %d``

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.
    argument: ``-step %f``

out_file: (a file name)
    argument: ``> %s``, position: -1

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
    argument: ``-frange %s``, position: 1

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.
    argument: ``-lrange %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

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.
    argument: ``-trace %G``

watson: (a boolean)
    Compute a LUT for the Watson PDF.
    argument: ``-watson``

acg: (a boolean)
    Compute a LUT for the ACG PDF.
    argument: ``-acg``

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

(continues on next page)

(continued from previous page)

```

bingham: (a boolean)
    Compute a LUT for the Bingham PDF. This is the default.
    argument: ``-bingham``
inversion: (an integer (int or long))
    Index of the inversion to use. The default is 1 (linear single
    tensor inversion).
    argument: ``-inversion %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.
    argument: ``-snr %f``

```

Outputs:

```

dtLUT: (an existing file name)
    Lookup Table

```

55.4.7 DTMetric

[Link to code](#)

Wraps the executable command dtshape.

Computes tensor metric statistics based on the eigenvalues $l1 \geq l2 \geq l3$ typically obtained from ComputeEigensystem.

The full list of statistics is:

- $\langle cl \rangle = (l1 - l2) / l1$, a measure of linearity
- $\langle cp \rangle = (l2 - l3) / l1$, a measure of planarity
- $\langle cs \rangle = l3 / l1$, a measure of isotropy with: $cl + cp + cs = 1$
- $\langle l1 \rangle$ = first eigenvalue
- $\langle l2 \rangle$ = second eigenvalue
- $\langle l3 \rangle$ = third eigenvalue
- $\langle tr \rangle = l1 + l2 + l3$
- $\langle md \rangle = tr / 3$
- $\langle rd \rangle = (l2 + l3) / 2$
- $\langle fa \rangle$ = 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 $l2$ and $l3$ i.e. $\sqrt{2 * [(l2 - \langle l \rangle)^2 + (l3 - \langle l \rangle)^2] / (l2^2 + l3^2)}$ with: $\langle l \rangle = (l2 + l3) / 2$

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]
metric: ('fa' or 'md' or 'rd' or 'l1' or 'l2' or 'l3' or 'tr' or 'ra'
        or '2dfa' or 'cl' or 'cp' or 'cs')

```

(continues on next page)

(continued from previous page)

```

    Specifies the metric to compute. Possible choices are: "fa", "md",
    "rd", "l1", "l2", "l3", "tr", "ra", "2dfa", "cl", "cp" or "cs".
    argument: ``-stat %s``
eigen_data: (an existing file name)
    voxel-order data filename
    argument: ``-inputfile %s``

[Optional]
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
    argument: ``-inputdatatype %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.
    argument: ``-header %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.
    argument: ``-outputfile %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
    argument: ``-outputdatatype %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

metric_stats: (an existing file name)
    Diffusion Tensor statistics of the chosen metric

```

55.4.8 ModelFit[Link to code](#)Wraps the executable 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
        argument: ``-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.
argument: ``-model %s``
```

(continues on next page)

(continued from previous page)

```

scheme_file: (an existing file name)
    Camino scheme file (b values / vectors, see camino.fsl2scheme)
    argument: ``-scheme_file %s``

[Optional]
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.
    argument: ``-tau %G``

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".
    argument: ``-inputdatatype %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.
    argument: ``-residualmap %s``

outputfile: (a file name)
    Filename of the output file.
    argument: ``-outputfile %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.
    argument: ``-fixedmod %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.
    argument: ``-bgthresh %G``

sigma: (a float)
    Specifies the standard deviation of the noise in the data. Required
    by the RESTORE algorithm.
    argument: ``-sigma %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

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.
    argument: ``-noisemap %s``

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.

```

(continues on next page)

(continued from previous page)

```

        argument: ``-csfthresh %G``
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.
        argument: ``-bgmask %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
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.
        argument: ``-fixedbvalue %s``
outlier: (an existing file name)
        Specifies the name of the file to contain the outlier map generated
        by the RESTORE algorithm.
        argument: ``-outliermap %s``
out_file: (a file name)
        argument: ``> %s``, position: -1

```

Outputs:

```

fitted_data: (an existing file name)
        output file of 4D volume in voxel order

```

55.4.9 PicoPDFs

[Link to code](#)Wraps the executable 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]
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.
        argument: ``-luts %s``
in_file: (an existing file name)
        voxel-order data filename
        argument: ``< %s``, position: 1

[Optional]
maxcomponents: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

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.
argument: ``-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.
argument: ``-numpds %d``
out_file: (a file name)
argument: ``> %s``, position: -1
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
inputmodel: ('dt' or 'multitensor' or 'pds', nipy default value:
dt)
input model type
argument: ``-inputmodel %s``, position: 2
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
argument: ``-pdf %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
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.
argument: ``-directmap``

```

Outputs:

```

pdfs: (an existing file name)
      path/name of 4D volume in voxel order

```

55.4.10 Track[Link to code](#)

Wraps the executable 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()

```

(continues on next page)

(continued from previous page)

```
>>> track.inputs.inputmodel = 'dt'
>>> track.inputs.in_file = 'data.Bfloat'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()
```

Inputs:

```
[Optional]
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.
    argument: ``-maxcomponents %d``

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.
    argument: ``-curveinterval %f``
    requires: curvethresh

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.
    argument: ``-tracker %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
    argument: ``-inputmodel %s``

environ: (a dictionary with keys which are a bytes or None or a 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
    argument: ``-gzip``

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.
    argument: ``-numpds %d``

data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
```

(continues on next page)

(continued from previous page)

```

    argument: ``-datadims %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.
    argument: ``-interpolator %s``
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    argument: ``-outputtracts %s``
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    argument: ``-voxeldims %s``
output_root: (a file name)
    root directory for output
    argument: ``-outputroot %s``, position: -1
inputdatatype: ('float' or 'double')
    input file type
    argument: ``-inputdatatype %s``
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.
    argument: ``-curvethresh %f``
out_file: (a file name)
    output data file
    argument: ``-outputfile %s``, position: -1
seed_file: (an existing file name)
    seed file
    argument: ``-seedfile %s``, position: 2
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    argument: ``-stepsize %f``
    requires: tracker
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``-ipthresh %f``
in_file: (an existing file name)
    input data file

```

(continues on next page)

(continued from previous page)

```

    argument: ``-inputfile %s``, position: 1
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.
    argument: ``-anisfile %s``
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    argument: ``-anisthresh %f``

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

55.4.11 TrackBallStick[Link to code](#)

Wraps the executable command track.

Performs streamline tractography using ball-stick fitted data

Example

```

>>> import nipype.interfaces.camino as cmon
>>> track = cmon.TrackBallStick()
>>> track.inputs.in_file = 'ballstickfit_data.Bfloat'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()

```

Inputs:

```

[Optional]
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.
    argument: ``-maxcomponents %d``
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.
    argument: ``-curveinterval %f``
    requires: curvethresh
tracker: ('fact' or 'euler' or 'rk4', nipype 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

```

(continues on next page)

(continued from previous page)

```

    of the step is determined by taking and averaging a series of
    partial steps.
    argument: ``-tracker %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
    argument: ``-inputmodel %s``
environ: (a dictionary with keys which are a bytes or None or a 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
    argument: ``-gzip``
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.
    argument: ``-numpds %d``
data_dims: (a list of from 3 to 3 items which are an integer (int or
           long))
    data dimensions in voxels
    argument: ``-datadims %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.
    argument: ``-interpolator %s``
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    argument: ``-outputtracts %s``
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    argument: ``-voxeldims %s``
output_root: (a file name)
    root directory for output
    argument: ``-outputroot %s``, position: -1
inputdatatype: ('float' or 'double')
    input file type
    argument: ``-inputdatatype %s``
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

```

(continues on next page)

(continued from previous page)

```

        streamline terminates.
        argument: ``-curvethresh %f``
out_file: (a file name)
        output data file
        argument: ``-outputfile %s``, position: -1
seed_file: (an existing file name)
        seed file
        argument: ``-seedfile %s``, position: 2
stepsize: (a float)
        Step size for EULER and RK4 tracking. The default is 1mm.
        argument: ``-stepsize %f``
        requires: tracker
args: (a unicode string)
        Additional parameters to the command
        argument: ``%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.
        argument: ``-ipthresh %f``
in_file: (an existing file name)
        input data file
        argument: ``-inputfile %s``, position: 1
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.
        argument: ``-anisfile %s``
anisthresh: (a float)
        Terminate fibres that enter a voxel with lower anisotropy than the
        threshold.
        argument: ``-anisthresh %f``

```

Outputs:

```

tracked: (an existing file name)
        output file containing reconstructed tracts

```

55.4.12 TrackBayesDirac[Link to code](#)Wraps the executable 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.
    argument: ``-schemefile %s``

[Optional]
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.
    argument: ``-maxcomponents %d``

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.
    argument: ``-curveinterval %f``
    requires: curvethresh

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.
    argument: ``-tracker %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
    argument: ``-inputmodel %s``

environ: (a dictionary with keys which are a bytes or None or a 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
    argument: ``-gzip``

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.
    argument: ``-curvepriorg %G``

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

```

(continues on next page)

(continued from previous page)

```

    computation time. The default is index 1, which gives 1922 points,
    index 0 gives 1082 points, index 2 gives 3002 points.
    argument: ``-pointset %s``
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    argument: ``-datadims %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.
    argument: ``-interpolator %s``
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    argument: ``-outputtracts %s``
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    argument: ``-voxeldims %s``
out_file: (a file name)
    output data file
    argument: ``-outputfile %s``, position: -1
inputdatatype: ('float' or 'double')
    input file type
    argument: ``-inputdatatype %s``
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.
    argument: ``-curvethresh %f``
output_root: (a file name)
    root directory for output
    argument: ``-outputroot %s``, position: -1
extpriordatatype: ('float' or 'double')
    Datatype of the prior image. The default is "double".
    argument: ``-extpriordatatype %s``
pdf: ('bingham' or 'watson' or 'acg')
    Specifies the model for PICO priors (not the curvature priors). The
    default is "bingham".
    argument: ``-pdf %s``
seed_file: (an existing file name)
    seed file
    argument: ``-seedfile %s``, position: 2
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.
    argument: ``-extpriorfile %s``

```

(continues on next page)

(continued from previous page)

```

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.
    argument: ``-curvepriork %G``
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    argument: ``-stepsize %f``
    requires: tracker
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``-ipthresh %f``
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.
    argument: ``-numpds %d``
datamodel: ('cylsymmmdt' or 'ballstick')
    Model of the data for Bayesian tracking. The default model is
    "cylsymmmdt", a diffusion tensor with cylindrical symmetry about e_1,
    ie L1 >= L_2 = L_3. The other model is "ballstick", the partial
    volume model (see ballstickfit).
    argument: ``-datamodel %s``
in_file: (an existing file name)
    input data file
    argument: ``-inputfile %s``, position: 1
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.
    argument: ``-anisfile %s``
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    argument: ``-anisthresh %f``
iterations: (an integer (int or long))
    Number of streamlines to generate at each seed point. The default is
    5000.
    argument: ``-iterations %d``

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```


55.4.13 TrackBedpostxDeter

[Link to code](#)

Wraps the executable 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_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 nipy.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
    argument: ``-bedpostxdir %s``

[Optional]
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.
    argument: ``-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.
    argument: ``-bedpostxminf %d``
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.
    argument: ``-curveinterval %f``
    requires: curvethresh
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.
```

(continues on next page)

(continued from previous page)

```

        argument: ``-tracker %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
            argument: ``-inputmodel %s``
environ: (a dictionary with keys which are a bytes or None or a 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
      argument: ``-gzip``
seed_file: (an existing file name)
           seed file
           argument: ``-seedfile %s``, position: 2
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.
        argument: ``-numpds %d``
data_dims: (a list of from 3 to 3 items which are an integer (int or
           long))
           data dimensions in voxels
           argument: ``-datadims %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.
              argument: ``-interpolator %s``
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.
             argument: ``-curvethresh %f``
voxel_dims: (a list of from 3 to 3 items which are a float)
            voxel dimensions in mm
            argument: ``-voxeldims %s``
output_root: (a file name)
             root directory for output
             argument: ``-outputroot %s``, position: -1
inputdatatype: ('float' or 'double')
              input file type
              argument: ``-inputdatatype %s``

```

(continues on next page)

(continued from previous page)

```

outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    argument: ``-outputtracts %s``
out_file: (a file name)
    output data file
    argument: ``-outputfile %s``, position: -1
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    argument: ``-stepsize %f``
    requires: tracker
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``-ipthresh %f``
in_file: (an existing file name)
    input data file
    argument: ``-inputfile %s``, position: 1
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.
    argument: ``-anisfile %s``
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    argument: ``-anisthresh %f``

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

55.4.14 TrackBedpostxProba[Link to code](#)

Wraps the executable 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 `X`th 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
    argument: ``-bedpostxdir %s``

[Optional]
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.
    argument: ``-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.
    argument: ``-bedpostxminf %d``
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.
    argument: ``-curveinterval %f``
    requires: curvethresh
tracker: ('fact' or 'euler' or 'rk4', nipyne 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.
    argument: ``-tracker %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',
    nipyne default value: dt)
    input model type
    argument: ``-inputmodel %s``
environ: (a dictionary with keys which are a 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
gzip: (a boolean)
```

(continues on next page)

(continued from previous page)

```

    save the output image in gzip format
    argument: ``-gzip``
seed_file: (an existing file name)
    seed file
    argument: ``-seedfile %s``, position: 2
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.
    argument: ``-numpds %d``
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    argument: ``-datadims %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.
    argument: ``-interpolator %s``
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.
    argument: ``-curvethresh %f``
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    argument: ``-voxeldims %s``
output_root: (a file name)
    root directory for output
    argument: ``-outputroot %s``, position: -1
inputdatatype: ('float' or 'double')
    input file type
    argument: ``-inputdatatype %s``
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    argument: ``-outputtracts %s``
out_file: (a file name)
    output data file
    argument: ``-outputfile %s``, position: -1
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    argument: ``-stepsize %f``
    requires: tracker
args: (a unicode string)
    Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

        argument: ``%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.
    argument: ``-ipthresh %f``
in_file: (an existing file name)
    input data file
    argument: ``-inputfile %s``, position: 1
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.
    argument: ``-anisfile %s``
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    argument: ``-anisthresh %f``
iterations: (an integer (int or long))
    Number of streamlines to generate at each seed point. The default is
    1.
    argument: ``-iterations %d``

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

55.4.15 TrackBootstrap[Link to code](#)Wraps the executable 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.
    argument: ``-bsdatafile %s``
scheme_file: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    The scheme file corresponding to the data being processed.
    argument: ``-schemefile %s``

[Optional]
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.
    argument: ``-maxcomponents %d``
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.
    argument: ``-curveinterval %f``
    requires: curvethresh
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.
    argument: ``-tracker %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
    argument: ``-inputmodel %s``
environ: (a dictionary with keys which are a bytes or None or a 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
    argument: ``-gzip``
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.
    argument: ``-numpds %d``
data_dims: (a list of from 3 to 3 items which are an integer (int or
    long))
    data dimensions in voxels
    argument: ``-datadims %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.

```

(continues on next page)

(continued from previous page)

```

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.
argument: ``-interpolator %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.
argument: ``-bgmask %s``

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.
argument: ``-curvethresh %f``

voxel_dims: (a list of from 3 to 3 items which are a float)
voxel dimensions in mm
argument: ``-voxeldims %s``

inversion: (an integer (int or long))
Tensor reconstruction algorithm for repetition bootstrapping.
Default is 1 (linear reconstruction, single tensor).
argument: ``-inversion %s``

out_file: (a file name)
output data file
argument: ``-outputfile %s``, position: -1

inputdatatype: ('float' or 'double')
input file type
argument: ``-inputdatatype %s``

outputtracts: ('float' or 'double' or 'oogl')
output tract file type
argument: ``-outputtracts %s``

output_root: (a file name)
root directory for output
argument: ``-outputroot %s``, position: -1

seed_file: (an existing file name)
seed file
argument: ``-seedfile %s``, position: 2

stepsize: (a float)
Step size for EULER and RK4 tracking. The default is 1mm.
argument: ``-stepsize %f``
requires: tracker

args: (a unicode string)
Additional parameters to the command
argument: ``%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

```

(continues on next page)

(continued from previous page)

```

        more than 80 degrees. Set this to -1.0 to disable curvature checking
        completely.
        argument: ``-ipthresh %f``
in_file: (an existing file name)
        input data file
        argument: ``-inputfile %s``, position: 1
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.
        argument: ``-anisfile %s``
anisthresh: (a float)
        Terminate fibres that enter a voxel with lower anisotropy than the
        threshold.
        argument: ``-anisthresh %f``
iterations: (an integer (int or long))
        Number of streamlines to generate at each seed point.
        argument: ``-iterations %d``

```

Outputs:

```

tracked: (an existing file name)
        output file containing reconstructed tracts

```

55.4.16 TrackDT

[Link to code](#)Wraps the executable command `track`.

Performs streamline tractography using tensor data

Example

```

>>> import nipype.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:

```

[Optional]
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.
        argument: ``-maxcomponents %d``
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.
        argument: ``-curveinterval %f``
requires: curvethresh
tracker: ('fact' or 'euler' or 'rk4', nipype default value: fact)
        The tracking algorithm controls streamlines are generated from the

```

(continues on next page)

(continued from previous page)

```

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.
argument: ``-tracker %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
argument: ``-inputmodel %s``
environ: (a dictionary with keys which are a bytes or None or a 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
      argument: ``-gzip``
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.
        argument: ``-numpds %d``
data_dims: (a list of from 3 to 3 items which are an integer (int or
           long))
           data dimensions in voxels
           argument: ``-datadims %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.
              argument: ``-interpolator %s``
outputtracts: ('float' or 'double' or 'oogl')
              output tract file type
              argument: ``-outputtracts %s``
voxel_dims: (a list of from 3 to 3 items which are a float)
            voxel dimensions in mm
            argument: ``-voxeldims %s``
output_root: (a file name)
            root directory for output

```

(continues on next page)

(continued from previous page)

```

        argument: ``-outputroot %s``, position: -1
inputdatatype: ('float' or 'double')
    input file type
    argument: ``-inputdatatype %s``
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.
    argument: ``-curvethresh %f``
out_file: (a file name)
    output data file
    argument: ``-outputfile %s``, position: -1
seed_file: (an existing file name)
    seed file
    argument: ``-seedfile %s``, position: 2
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    argument: ``-stepsize %f``
    requires: tracker
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``-ipthresh %f``
in_file: (an existing file name)
    input data file
    argument: ``-inputfile %s``, position: 1
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.
    argument: ``-anisfile %s``
anisthresh: (a float)
    Terminate fibres that enter a voxel with lower anisotropy than the
    threshold.
    argument: ``-anisthresh %f``

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

55.4.17 TrackPICO[Link to code](#)Wraps the executable command `track`.

Performs streamline tractography using the Probabilistic Index of Connectivity (PICO) algorithm

Example

```
>>> import nipyype.interfaces.camino as cmon
>>> track = cmon.TrackPICO()
>>> track.inputs.in_file = 'pdfs.Bfloat'
>>> track.inputs.seed_file = 'seed_mask.nii'
>>> track.run()
```

Inputs:

```
[Optional]
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.
    argument: ``-maxcomponents %d``
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.
    argument: ``-curveinterval %f``
    requires: curvethresh
tracker: ('fact' or 'euler' or 'rk4', nipyype 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.
    argument: ``-tracker %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
    argument: ``-inputmodel %s``
environ: (a dictionary with keys which are a 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
    argument: ``-gzip``
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.
    argument: ``-numpds %d``
data_dims: (a list of from 3 to 3 items which are an integer (int or
```

(continues on next page)

(continued from previous page)

```

        long))
        data dimensions in voxels
        argument: ``-datadims %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.
    argument: ``-interpolator %s``
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.
    argument: ``-curvethresh %f``
voxel_dims: (a list of from 3 to 3 items which are a float)
    voxel dimensions in mm
    argument: ``-voxeldims %s``
out_file: (a file name)
    output data file
    argument: ``-outputfile %s``, position: -1
inputdatatype: ('float' or 'double')
    input file type
    argument: ``-inputdatatype %s``
outputtracts: ('float' or 'double' or 'oogl')
    output tract file type
    argument: ``-outputtracts %s``
output_root: (a file name)
    root directory for output
    argument: ``-outputroot %s``, position: -1
pdf: ('bingham' or 'watson' or 'acg')
    Specifies the model for PICO parameters. The default is "bingham.
    argument: ``-pdf %s``
seed_file: (an existing file name)
    seed file
    argument: ``-seedfile %s``, position: 2
stepsize: (a float)
    Step size for EULER and RK4 tracking. The default is 1mm.
    argument: ``-stepsize %f``
    requires: tracker
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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

```

(continues on next page)

(continued from previous page)

```

        more than 80 degrees. Set this to -1.0 to disable curvature checking
        completely.
        argument: ``-ipthresh %f``
in_file: (an existing file name)
        input data file
        argument: ``-inputfile %s``, position: 1
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.
        argument: ``-anisfile %s``
anisthresh: (a float)
        Terminate fibres that enter a voxel with lower anisotropy than the
        threshold.
        argument: ``-anisthresh %f``
iterations: (an integer (int or long))
        Number of streamlines to generate at each seed point. The default is
        5000.
        argument: ``-iterations %d``

```

Outputs:

```

tracked: (an existing file name)
        output file containing reconstructed tracts

```

55.5 interfaces.camino.odf

55.5.1 LinRecon

[Link to code](#)

Wraps the executable 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 p_1, \dots, p_R 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]
qball_mat: (an existing file name)
    Linear transformation matrix.
    argument: ``%s``, position: 3
in_file: (an existing file name)
    voxel-order data filename
    argument: ``%s``, position: 1
scheme_file: (an existing file name)
    Specifies the scheme file for the diffusion MRI data
    argument: ``%s``, position: 2

[Optional]
normalize: (a boolean)
    Normalize the measurements and discard the zero measurements before
    the linear transform.
    argument: ``-normalize``
out_file: (a file name)
    argument: ``> %s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
bgmask: (an existing file name)
    background mask
    argument: ``-bgmask %s``
log: (a boolean)
    Transform the log measurements rather than the measurements
    themselves
    argument: ``-log``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

recon_data: (an existing file name)
    Transformed data

```

55.5.2 MESD[Link to code](#)

Wraps the executable 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, $\ln(A^{\text{star}}(0))$, 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 $\langle X \rangle = N$, 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 $\langle X \rangle$ here specifies the number of lambda parameters. To obtain speed-up, set $\langle X \rangle < N$; complexity become $O(\langle X \rangle^2)$ rather than $O(N^2)$. Note that $\langle X \rangle$ 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 $\langle X \rangle$ is low `-fastmesd` is worth using to get even more speed-up.

The choice of $\langle X \rangle$ 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 $\langle X \rangle = 16$ 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]
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
    argument: ``-filter %s``, position: 2
inverter_param: (a float)
```

(continues on next page)

(continued from previous page)

```

    Parameter associated with the inverter. Cf. inverter description
    formore information.
    argument: ``%f``, position: 3
in_file: (an existing file name)
    voxel-order data filename
    argument: ``-inputfile %s``, position: 1
scheme_file: (an existing file name)
    Specifies the scheme file for the diffusion MRI data
    argument: ``-schemefile %s``

[Optional]
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".
    argument: ``-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".
    argument: ``-mepointset %d``
fastmesd: (a boolean)
    Turns off numerical integration checks and fixes the integration
    point set size at that of the index specified by -basepointset..
    argument: ``-fastmesd``
    requires: mepointset
out_file: (a file name)
    argument: ``> %s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
bgmask: (an existing file name)
    background mask
    argument: ``-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

```

Outputs:

```

mesd_data: (an existing file name)
    MESD data

```

55.5.3 QBallIMX[Link to code](#)

Wraps the executable 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 nipy.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
    argument: ``-schemefile %s``

[Optional]
smoothingsigma: (a float)
    Specific to rbf. Sets the width of the smoothing basis functions.
    The default value is 0.1309 (7.5 degrees).
    argument: ``-smoothingsigma %f``
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.
    argument: ``-rbfpointset %d``
basistype: ('rbf' or 'sh', nipy default value: rbf)
    Basis function type. "rbf" to use radial basis functions "sh" to use
    spherical harmonics
    argument: ``-basistype %s``
rbfsigma: (a float)
    Specific to rbf. Sets the width of the interpolating basis
    functions. The default value is 0.2618 (15 degrees).
    argument: ``-rbfsigma %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
order: (an integer (int or long))
    Specific to sh. Maximum order of the spherical harmonic series.
    Default is 4.
```

(continues on next page)

(continued from previous page)

```

argument: ``-order %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
out_file: (a file name)
argument: ``> %s``, position: -1

```

Outputs:

```

qmat: (an existing file name)
      Q-Ball reconstruction matrix

```

55.5.4 SFPeaks

[Link to code](#)

Wraps the executable 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.)
- $\text{mean}(f)$.
- $\text{std}(f)$.
- 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 nipyne.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
    argument: ``-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" file"
    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.
    argument: ``-inputmodel %s``

[Optional]
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).
    argument: ``-pdthresh %f``
out_file: (a file name)
    argument: ``> %s``, position: -1
noconsistencycheck: (a boolean)
    Turns off the consistency check. The output shows all consistencies
    as true.
    argument: ``-noconsistencycheck``
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.
    argument: ``-density %d``
order: (an integer (int or long))
    Specific to sh. Maximum order of the spherical harmonic series.
```

(continues on next page)

(continued from previous page)

```

    argument: ``-order %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', nipyne default value: {})
    Environment variables
searchradius: (a float)
    The search radius in the peak finding algorithm. The default is 0.4
    (cf. "density")
    argument: ``-searchradius %f``
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" file"
    attribute.
    argument: ``-mepointset %d``
numpds: (an integer (int or long))
    The largest number of peak directions to output in each voxel.
    argument: ``-numpds %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
scheme_file: (an existing file name)
    Specific to maxent. Specifies the scheme file.
    argument: ``%s``
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.
    argument: ``-pointset %d``
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.
    argument: ``-stdsfrommean %f``
rbfpntset: (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.
    argument: ``-rbfpntset %d``

```

Outputs:

```

peaks: (an existing file name)
    Peaks of the spherical functions.

```

55.6 interfaces.camino.utils

55.6.1 ImageStats

[Link to code](#)Wraps the executable 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.
    argument: ``-images %s``, position: -1
stat: ('min' or 'max' or 'mean' or 'median' or 'sum' or 'std' or
      'var')
    The statistic to compute.
    argument: ``-stat %s``
output_root: (a file name)
    Filename root prepended onto the names of the output files. The
    extension will be determined from the input.
    argument: ``-outputroot %s``

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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.
    argument: ``-outputdatatype %s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
out_file: (an existing file name)
    Path of the file computed with the statistic chosen
```

56.1 interfaces.camino2trackvis.convert

56.1.1 Camino2Trackvis

[Link to code](#)

Wraps the executable 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]
in_file: (an existing file name)
    The input .Bfloat (camino) file.
    argument: ``-i %s``, position: 1
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.
    argument: ``--voxel-order %s``, position: 6
voxel_dims: (a list of from 3 to 3 items which are a float)
```

(continues on next page)

(continued from previous page)

```

    Three comma-separated numbers giving the size of each voxel in mm.
    argument: ``-x %s``, position: 5
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.
    argument: ``-d %s``, position: 4

[Optional]
min_length: (a float)
    The minimum length of tracts to output
    argument: ``-l %d``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
nifti_file: (an existing file name)
    Read coordinate system from a NIfTI file.
    argument: ``--nifti %s``, 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', nipy default value: {})
    Environment variables
out_file: (a file name)
    The filename to which to write the .trk (trackvis) file.
    argument: ``-o %s``, position: 2

```

Outputs:

```

trackvis: (an existing file name)
    The filename to which to write the .trk (trackvis) file.

```

56.1.2 Trackvis2Camino[Link to code](#)Wraps the executable command `trackvis_to_camino`.**Inputs:**

```

[Mandatory]
in_file: (an existing file name)
    The input .trk (trackvis) file.
    argument: ``-i %s``, position: 1

[Optional]
append_file: (an existing file name)
    A file to which the append the .Bfloat data.
    argument: ``-a %s``, position: 2
out_file: (a file name)
    The filename to which to write the .Bfloat (camino).
    argument: ``-o %s``, position: 2
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:


```
camino: (an existing file name)
        The filename to which to write the .Bfloat (camino).
```


57.1 interfaces.cmtk.base

57.1.1 CFFBaseInterface

[Link to code](#)

Inputs:

None

Outputs:

None

57.2 interfaces.cmtk.cmtk

57.2.1 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]
roi_file: (an existing file name)
    Freesurfer aparc+aseg file
resolution_network_file: (an existing file name)
    Parcellation files from Connectome Mapping Toolkit
```

(continues on next page)

(continued from previous page)

```

tract_file: (an existing file name)
    Trackvis tract file

[Optional]
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_endpoint_array_name: (a file name)
    Name for the generated endpoint arrays
out_mean_fiber_length_matrix_mat_file: (a file name)
    Matlab matrix describing the mean fiber lengths between each node.
out_intersection_matrix_mat_file: (a file name)
    Matlab connectivity matrix if all region/fiber intersections are
    counted.
out_median_fiber_length_matrix_mat_file: (a file name)
    Matlab matrix describing the mean fiber lengths between each node.
count_region_intersections: (a boolean, nipy default value: False)
    Counts all of the fiber-region traversals in the connectivity matrix
    (requires significantly more computational time)
out_fiber_length_std_matrix_mat_file: (a file name)
    Matlab matrix describing the deviation in fiber lengths connecting
    each node.

```

Outputs:

```

endpoint_file: (an existing file name)
    Saved Numpy array with the endpoints of each fiber
fiber_label_file: (an existing file name)
    Saved Numpy array with the labels for each fiber
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.
fiber_length_std_matrix_mat_file: (an existing file name)
    Matlab matrix describing the deviation in fiber lengths connecting
    each node.
stats_file: (an existing file name)
    Saved Matlab .mat file with the number of fibers saved at each stage
endpoint_file_mm: (an existing file name)
    Saved Numpy array with the endpoints of each fiber (in millimeters)
median_fiber_length_matrix_mat_file: (an existing file name)
    Matlab matrix describing the median fiber lengths between each node.
matrix_file: (an existing file name)
    NetworkX graph describing the connectivity
filtered_tractography: (an existing file name)
    TrackVis file containing only those fibers originate in one and
    terminate in another region
fiber_length_file: (an existing file name)
    Saved Numpy array with the lengths of each fiber
filtered_tractographies: (a list of items which are an existing file
    name)
intersection_matrix_mat_file: (an existing file name)
    Matlab matrix describing the mean fiber lengths between each node.
fiber_labels_noorphans: (an existing file name)
    Saved Numpy array with the labels for each non-orphan fiber
filtered_tractography_by_intersections: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    TrackVis file containing all fibers which connect two regions
matlab_matrix_files: (a list of items which are an existing file
    name)
intersection_matrix_file: (an existing file name)
    NetworkX graph describing the connectivity
matrix_files: (a list of items which are an existing file name)

```

57.2.2 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]
roi_file: (an existing file name)
    Region of interest file
resolution_network_file: (an existing file name)
    Parcellation file from Connectome Mapping Toolkit

[Optional]
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.

```

57.2.3 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")

```

(continues on next page)

(continued from previous page)

```
>>> labelDict = pickle.load(file)
>>> labelDict
```

Inputs:

```
[Mandatory]
aparc_aseg_file: (an existing file name)
    Freesurfer aparc+aseg file

[Optional]
out_dict_file: (a file name)
    Label dictionary saved in Pickle format
use_freesurfer_LUT: (a boolean)
    Boolean value; Set to True to use default Freesurfer LUT, False for
    custom LUT
mutually_exclusive: LUT_file
out_roi_file: (a file name)
    Region of Interest file for connectivity mapping
freesurfer_dir: (a directory name)
    Freesurfer main directory
    requires: use_freesurfer_LUT
LUT_file: (an existing file name)
    Custom lookup table (cf. FreeSurferColorLUT.txt)
mutually_exclusive: use_freesurfer_LUT
```

Outputs:

```
roi_file: (a file name)
    Region of Interest file for connectivity mapping
dict_file: (a file name)
    Label dictionary saved in Pickle format
```

57.2.4 cmat()[Link to code](#)

Create the connection matrix for each resolution using fibers and ROIs.

57.2.5 create_allpoints_cmat()[Link to code](#)

Create the intersection arrays for each fiber

57.2.6 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 millimeter coordinates

57.2.7 create_nodes()[Link to code](#)**57.2.8 get_connectivity_matrix()**[Link to code](#)

57.2.9 `get_rois_crossed()`

[Link to code](#)

57.2.10 `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])

```

57.2.11 `save_fibers()`

[Link to code](#)

Stores a new trackvis file fname using only given indices

57.3 `interfaces.cmtk.convert`

57.3.1 `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']

```

(continues on next page)

(continued from previous page)

```
>>> cvt.inputs.tract_files = ['streamlines.trk']
>>> cvt.inputs.gpickled_networks = ['network0.gpickle']
>>> cvt.run()
```

Inputs:

```
[Optional]
script_files: (a list of items which are an existing file name)
               list of script files to include
tract_files: (a list of items which are an existing file name)
               list of Trackvis fiber files
description: (a unicode string, nipyne default value: Created with
               the Nipyne CFF converter)
               Description
out_file: (a file name, nipyne default value: connectome.cff)
           Output connectome file
email: (a unicode string)
        Email address
relation: (a unicode string)
           Relation
timeseries_files: (a list of items which are an existing file name)
                   list of HDF5 timeseries files
publisher: (a unicode string)
            Publisher
gpickled_networks: (a list of items which are an existing file name)
                    list of gpickled Networkx graphs
title: (a unicode string)
        Connectome Title
gifti_labels: (a list of items which are an existing file name)
               list of GIFTI labels
data_files: (a list of items which are an existing file name)
              list of external data files (i.e. Numpy, HD5, XML)
rights: (a unicode string)
         Rights
graphml_networks: (a list of items which are an existing file name)
                   list of graphML networks
creator: (a unicode string)
          Creator
gifti_surfaces: (a list of items which are an existing file name)
                 list of GIFTI surfaces
license: (a unicode string)
          License
species: (a unicode string, nipyne default value: Homo sapiens)
          Species
references: (a unicode string)
            References
nifti_volumes: (a list of items which are an existing file name)
                list of NIFTI volumes
```

Outputs:

```
connectome_file: (an existing file name)
                  Output connectome file
```

57.3.2 MergeCNetworks

[Link to code](#)

Merges networks from multiple CFF files into one new CFF file.

Example

```
>>> import nipyne.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]
out_file: (a file name, nipyne 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
```

57.4 interfaces.cmtk.nbs

57.4.1 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 nipyne.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]
threshold: (a float, nipyne default value: 3)
           T-statistic threshold
node_position_network: (a file name)
                      An optional network used to position the nodes for the output
                      networks
t_tail: ('left' or 'right' or 'both', nipyne default value: left)
```

(continues on next page)

(continued from previous page)

```

    Can be one of "left", "right", or "both"
out_nbs_network: (a file name)
    Output network with edges identified by the NBS
number_of_permutations: (an integer (int or long), nipy default
    value: 1000)
    Number of permutations to perform
out_nbs_pval_network: (a file name)
    Output network with p-values to weight the edges identified by the
    NBS
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.

```

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

```

57.4.2 ntwks_to_matrices()[Link to code](#)**57.5 interfaces.cmtk.nx****57.5.1 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 nipy.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]
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

```

(continues on next page)

(continued from previous page)

```

        necessary, but if included, the interface will output the
        statistical maps as networkx graphs.
group_id: (a unicode string, nipy default value: group1)
        ID for group
out_gexf_groupavg: (a file name)
        Average network saved as a .gexf file

```

Outputs:

```

gexf_groupavg: (a file name)
        Average network saved as a .gexf file
matlab_groupavgs: (a list of items which are a file name)
gpickled_groupavg: (a file name)
        Average network saved as a NetworkX .pck

```

57.5.2 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
treat_as_weighted_graph: (a boolean, nipy default value: True)
        Some network metrics can be calculated while considering only a
        binarized version of the graph
out_k_shell: (a file name, nipy default value: k_shell)
        Computed k-shell network stored as a NetworkX pickle.
out_pickled_extra_measures: (a file name, nipy default value:
        extra_measures)
        Network measures for group 1 that return dictionaries stored as a
        Pickle.
out_node_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_edge_metrics_matlab: (a file name)
        Output edge metrics in MATLAB .mat format

```

(continues on next page)

(continued from previous page)

```
out_global_metrics_matlab: (a file name)
    Output node metrics in MATLAB .mat format
```

Outputs:

```
edge_measures_matlab: (a file name)
    Output edge metrics in MATLAB .mat format
gpickled_network_files: (a list of items which are a file name)
k_networks: (a list of items which are a file name)
pickled_extra_measures: (a file name)
    Network measures for the group that return dictionaries, stored as a
    Pickle.
k_core: (a file name)
    Computed k-core network stored as a NetworkX pickle.
matlab_dict_measures: (a list of items which are a file name)
node_measures_matlab: (a file name)
    Output node metrics in MATLAB .mat format
global_measures_matlab: (a file name)
    Output global metrics in MATLAB .mat format
k_shell: (a file name)
    Computed k-shell network stored as a NetworkX pickle.
k_crust: (a file name)
    Computed k-crust network stored as a NetworkX pickle.
node_measure_networks: (a list of items which are a file name)
edge_measure_networks: (a list of items which are a file name)
matlab_matrix_files: (a list of items which are a file name)
```

57.5.3 add_dicts_by_key()

[Link to code](#)

Combines two dictionaries and adds the values for those keys that are shared

57.5.4 add_edge_data()

[Link to code](#)

57.5.5 add_node_data()

[Link to code](#)

57.5.6 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

57.5.7 compute_dict_measures()

[Link to code](#)

Returns a dictionary

57.5.8 compute_edge_measures()

[Link to code](#)

These return edge-based measures

57.5.9 compute_network_measures()

[Link to code](#)

57.5.10 compute_node_measures()

[Link to code](#)

These return node-based measures

57.5.11 compute_singlevalued_measures()

[Link to code](#)

Returns a single value per network

57.5.12 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

57.5.13 read_unknown_ntwk()

[Link to code](#)

57.5.14 remove_all_edges()

[Link to code](#)

57.6 interfaces.cmtk.parcellation

57.6.1 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]
subjects_dir: (an existing directory name)
```

(continues on next page)

(continued from previous page)

```

    Freesurfer subjects directory
out_roi_file: (a file name)
    Region of Interest file for connectivity mapping
freesurfer_dir: (an existing directory name)
    Freesurfer main directory
dilation: (a boolean, nipy default value: False)
    Dilate cortical parcels? Useful for fMRI connectivity
parcellation_name: ('scale33' or 'scale60' or 'scale125' or
    'scale250' or 'scale500', nipy default value: scale500)

```

Outputs:

```

roiv_file: (a file name)
    Region of Interest file for fMRI connectivity mapping
cc_unknown_file: (an existing file name)
    Image file with regions labelled as unknown cortical structures
roi_file_in_structural_space: (an existing file name)
    ROI image resliced to the dimensions of the original structural
    image
white_matter_mask_file: (an existing file name)
    White matter mask file
aseg_file: (an existing file name)
    Automated segmentation file converted from Freesurfer "subjects"
    directory
roi_file: (an existing file name)
    Region of Interest file for connectivity mapping
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

```

57.6.2 create_annot_label()[Link to code](#)**57.6.3 create_roi()**[Link to code](#)

Creates the ROI_%.nii.gz files using the given parcellation information from networks. Iteratively create volume.

57.6.4 create_wm_mask()[Link to code](#)**57.6.5 crop_and_move_datasets()**[Link to code](#)**57.6.6 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

58.1 interfaces.diffusion_toolkit.dti

58.1.1 DTIRecon

[Link to code](#)

Wraps the executable command `dti_recon`.

Use `dti_recon` to generate tensors and other maps

Inputs:

```

[Mandatory]
DWI: (an existing file name)
    Input diffusion volume
    argument: ``%s``, position: 1
bvals: (an existing file name)
    b values file
bvecs: (an existing file name)
    b vectors file
    argument: ``-gm %s``

[Optional]
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
    argument: ``-oc``
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
    argument: ``-iop %f``

```

(continues on next page)

(continued from previous page)

```

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
output_type: ('nii' or 'analyze' or 'nii.gz', nipy default
    value: nii)
    output file type
    argument: ``-ot %s``
out_prefix: (a unicode string, nipy default value: dti)
    Output file prefix
    argument: ``%s``, position: 2
n_averages: (an integer (int or long))
    Number of averages
    argument: ``-nex %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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.
    argument: ``-b0_th``

```

Outputs:

```

L3: (an existing file name)
V2: (an existing file name)
FA: (an existing file name)
FA_color: (an existing file name)
V1: (an existing file name)
tensor: (an existing file name)
exp: (an existing file name)
L1: (an existing file name)
B0: (an existing file name)
L2: (an existing file name)
V3: (an existing file name)
ADC: (an existing file name)

```

58.1.2 DTITracker[Link to code](#)Wraps the executable command `dti_tracker`.**Inputs:**

```

[Mandatory]
mask1_file: (a file name)
    first mask image
    argument: ``-m %s``, position: 2

[Optional]
angle_threshold_weight: (a float)
    set angle threshold weighting factor. weighting will be applied
    on top of the angle_threshold
    argument: ``-atw %f``
swap_zx: (a boolean)
    swap x & z vectors while tracking

```

(continues on next page)

(continued from previous page)

```

        argument: ``-szx``
tensor_file: (an existing file name)
    reconstructed tensor file
invert_z: (a boolean)
    invert z component of the vector
    argument: ``-iz``
swap_yz: (a boolean)
    swap y & z vectors while tracking
    argument: ``-syz``
output_mask: (a file name)
    output a binary mask file in analyze format
    argument: ``-om %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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'
    argument: ``-it %s``
mask1_threshold: (a float)
    threshold value for the first mask image, if not given, the program
    will try automatically find the threshold
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
    argument: ``-s %s``
invert_x: (a boolean)
    invert x component of the vector
    argument: ``-ix``
output_file: (a file name, nipy default value: tracks.trk)
    argument: ``%s``, position: 1
input_data_prefix: (a unicode string, nipy default value: dti)
    for internal naming use only
    argument: ``%s``, position: 0
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
    argument: ``-l %f``
mask2_threshold: (a float)
    threshold value for the second mask image, if not given, the program
    will try automatically find the threshold
primary_vector: ('v2' or 'v3')
    which vector to use for fibre tracking: v2 or v3. If not set use v1
    argument: ``-v %s``
angle_threshold: (a float)
    set angle threshold. default value is 35 degree
    argument: ``-at %f``
args: (a unicode string)
    Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
mask2_file: (a file name)
    second mask image
    argument: ``-m2 %s``, position: 4
swap_xy: (a boolean)
    swap x & y vectors while tracking
    argument: ``-sxy``
invert_y: (a boolean)
    invert y component of the vector
    argument: ``-iy``
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
    argument: ``-rseed %d``

```

Outputs:

```

track_file: (an existing file name)
mask_file: (an existing file name)

```

58.2 interfaces.diffusion_toolkit.odf

58.2.1 HARDIMat

[Link to code](#)Wraps the executable 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
    argument: ``%s``, position: 1

[Optional]
out_file: (a file name, nipy default value: recon_mat.dat)
    output matrix file
    argument: ``%s``, position: 2
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
    argument: ``-oc``
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
    argument: ``-ref %s``
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

Additional parameters to the command
argument: ``%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
    argument: ``-iop %f``
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
    argument: ``-info %s``
order: (an integer (int or long))
    maximum order of spherical harmonics. must be even number. default
    is 4
    argument: ``-order %s``
environ: (a dictionary with keys which are a 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
odf_file: (an existing file name)
    filename that contains the reconstruction points on a HEMI-sphere.
    use the pre-set 181 points by default
    argument: ``-odf %s``

```

Outputs:

```

out_file: (an existing file name)
    output matrix file

```

58.2.2 ODFRecon[Link to code](#)Wraps the executable command `odf_recon`.Use `odf_recon` to generate tensors and other maps**Inputs:**

```

[Mandatory]
n_output_directions: (an integer (int or long))
    Number of output directions
    argument: ``%s``, position: 3
DWI: (an existing file name)
    Input raw data
    argument: ``%s``, position: 1
matrix: (an existing file name)
    use given file as reconstruction matrix.
    argument: ``-mat %s``
n_directions: (an integer (int or long))
    Number of directions
    argument: ``%s``, position: 2

```

(continues on next page)

(continued from previous page)

```

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
    argument: ``-b0 %s``

[Optional]
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
    argument: ``-s %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
    argument: ``-oc``
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
    argument: ``-iop %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
output_type: ('nii' or 'analyze' or 'nii' or 'nii.gz', nipy default
    value: nii)
    output file type
    argument: ``-ot %s``
subtract_background: (a boolean)
    subtract the background value before reconstruction
    argument: ``-bg``
out_prefix: (a unicode string, nipy default value: odf)
    Output file prefix
    argument: ``%s``, position: 4
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
dsi: (a boolean)
    indicates that the data is dsi
    argument: ``-dsi``
output_entropy: (a boolean)
    output entropy map
    argument: ``-oe``
filter: (a boolean)
    apply a filter (e.g. high pass) to the raw image
    argument: ``-f``

```

Outputs:

```

ODF: (an existing file name)
DWI: (an existing file name)
B0: (an existing file name)
max: (an existing file name)
entropy: (a file name)

```

58.2.3 ODFTracker

[Link to code](#)

Wraps the executable 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
    argument: ``-m %s``, position: 2
max: (an existing file name)

[Optional]
swap_zx: (a boolean)
    swap x and z vectors while tracking
    argument: ``-szx``
runge_kutta2: (a boolean)
    use 2nd order runge-kutta method for tracking.
    default tracking method is non-interpolate streamline
    argument: ``-rk2``
angle_threshold: (a float)
    set angle threshold. default value is 35 degree for
    default tracking method and 25 for rk2
    argument: ``-at %f``
mask2_file: (a file name)
    second mask image
    argument: ``-m2 %s``, position: 4
invert_z: (a boolean)
    invert z component of the vector
    argument: ``-iz``
swap_yz: (a boolean)
    swap y and z vectors while tracking
    argument: ``-syz``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
mask1_threshold: (a float)
    threshold value for the first mask image, if not given, the program
    will try automatically find the threshold
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

```

(continues on next page)

(continued from previous page)

```

    argument: ``-iop %f``
out_file: (a file name, nipy default value: tracks.trk)
    output track file
    argument: ``%s``, position: 1
invert_x: (a boolean)
    invert x component of the vector
    argument: ``-ix``
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)
    argument: ``-limit %d``
input_data_prefix: (a unicode string, nipy default value: odf)
    recon data prefix
    argument: ``%s``, position: 0
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
    argument: ``-vorder %s``
step_length: (a float)
    set step length, in the unit of minimum voxel size.
    default value is 0.1.
    argument: ``-l %f``
mask2_threshold: (a float)
    threshold value for the second mask image, if not given, the program
    will try automatically find the threshold
slice_order: (an integer (int or long))
    set the slice order. 1 means normal, -1 means reversed. default
    value is 1
    argument: ``-sorder %d``
disc: (a boolean)
    use disc tracking
    argument: ``-disc``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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.
    argument: ``-dsi``
input_output_type: ('nii' or 'analyze' or 'nii.gz', nipy
    default value: nii)
    input and output file type
    argument: ``-it %s``
swap_xy: (a boolean)
    swap x and y vectors while tracking

```

(continues on next page)

(continued from previous page)

```

        argument: ``-sxy``
invert_y: (a boolean)
        invert y component of the vector
        argument: ``-iy``
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
        argument: ``-rseed %s``

```

Outputs:

```

track_file: (an existing file name)
        output track file

```

58.3 interfaces.diffusion_toolkit.postproc

58.3.1 SplineFilter

[Link to code](#)

Wraps the executable 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 nipype.interfaces.diffusion_toolkit as dtk
>>> filt = dtk.SplineFilter()
>>> filt.inputs.track_file = 'tracks.trk'
>>> filt.inputs.step_length = 0.5
>>> filt.run()

```

Inputs:

```

[Mandatory]
track_file: (an existing file name)
        file containing tracks to be filtered
        argument: ``%s``, position: 0
step_length: (a float)
        in the unit of minimum voxel size
        argument: ``%f``, position: 1

[Optional]
environ: (a dictionary with keys which are a 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
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
output_file: (a file name, nipype default value: spline_tracks.trk)
        target file for smoothed tracks
        argument: ``%s``, position: 2

```

Outputs:

```
smoothed_track_file: (an existing file name)
```

58.3.2 TrackMerge

[Link to code](#)

Wraps the executable 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
              argument: ``%s...``, position: 0

[Optional]
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
output_file: (a file name, nipy default value: merged_tracks.trk)
              target file for merged tracks
              argument: ``%s``, position: -1
```

Outputs:

```
track_file: (an existing file name)
```


59.1 interfaces.dipy.anisotropic_power

59.1.1 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]
mask_file: (an existing file name)
           An optional brain mask
out_prefix: (a unicode string)
           output prefix for file names
b0_thres: (an integer (int or long), nipype default value: 700)
          b0 threshold
```

Outputs:

```
out_file: (an existing file name)
```

59.2 interfaces.dipy.base

59.2.1 DipyBaseInterface

[Link to code](#)

A base interface for py:mod:*dipy* computations

Inputs:

```
None
```

Outputs:

```
None
```

59.2.2 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]
out_prefix: (a unicode string)
            output prefix for file names
b0_thres: (an integer (int or long), nipy default value: 700)
          b0 threshold
```

Outputs:

```
None
```

59.3 interfaces.dipy.preprocess

59.3.1 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]
snr: (a float)
    manually set an SNR
noise_mask: (an existing file name)
    mask in which the standard deviation of noise will be computed
patch_radius: (an integer (int or long), nipy default value: 1)
    patch radius
block_radius: (an integer (int or long), nipy default value: 5)
    block radius
in_mask: (an existing file name)
    brain mask
signal_mask: (an existing file name)
    mask in which the mean signal will be computed
```

Outputs:

```
out_file: (an existing file name)
```

59.3.2 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)
```

59.3.3 nlmeans_proxy()

[Link to code](#)

Uses non-local means to denoise 4D datasets

59.3.4 resample_proxy()

[Link to code](#)

Performs regridding of an image to set isotropic voxel sizes using dipy.

59.4 interfaces.dipy.reconstruction

59.4.1 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]
save_fods: (a boolean, nipy default value: True)
           save fODFs in file
response: (an existing file name)
           single fiber estimated response
out_prefix: (a unicode string)
            output prefix for file names
b0_thres: (an integer (int or long), nipy default value: 700)
           b0 threshold
sh_order: (an integer (int or long), nipy default value: 8)
           maximal shperical harmonics order
out_fods: (a file name)
           fODFs output file name
in_mask: (an existing file name)
          input mask in which compute tensors
```

Outputs:

```
out_fods: (a file name)
          fODFs output file name
```

(continues on next page)

(continued from previous page)

```
model: (a file name)
        Python pickled object of the CSD model fitted.
```

59.4.2 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_file: (an existing file name)
         input diffusion data
in_evals: (an existing file name)
          input eigenvalues file

[Optional]
response: (a file name, nipy default value: response.txt)
          the output response file
b0_thres: (an integer (int or long), nipy default value: 700)
          b0 threshold
recursive: (a boolean)
           use the recursive response estimator from dipy
           mutually_exclusive: auto
in_mask: (an existing file name)
         input mask in which we find single fibers
out_prefix: (a unicode string)
            output prefix for file names
out_mask: (a file name, nipy default value: wm_mask.nii.gz)
           computed wm mask
auto: (a boolean)
      use the auto_response estimator from dipy
      mutually_exclusive: recursive
roi_radius: (an integer (int or long), nipy default value: 10)
            ROI radius to be used in auto_response
fa_thresh: (a float, nipy default value: 0.7)
            FA threshold
```

Outputs:

```

response: (an existing file name)
           the response file
out_mask: (an existing file name)
           output wm mask

```

59.4.3 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]
out_prefix: (a unicode string)
            output prefix for file names
noise_mask: (an existing file name)
            input mask in which compute noise variance
b0_thres: (an integer (int or long), nipy default value: 700)
          b0 threshold
in_mask: (an existing file name)
          input mask in which compute tensors

```

Outputs:

```

rd: (a file name)
    output radial diffusivity (RD) 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
trace: (a file name)
    output the tensor trace map computed from the fitted DTI
evecs: (a file name)
    output the eigenvectors of the fitted DTI
evals: (a file name)
    output the eigenvalues of the fitted DTI
fa: (a file name)
    output fractional anisotropy (FA) map computed from the fitted DTI

```

59.5 interfaces.dipy.simulate

59.5.1 SimulateMultiTensor

[Link to code](#)

Interface to MultiTensor model simulator in dipy http://nipy.org/dipy/examples_built/simulate_multi_tensor.html

Example

```
>>> import nipy.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]
in_dirs: (a list of items which are an existing file name)
         list of fibers (principal directions)
baseline: (an existing file name)
         baseline T2 signal
in_vfms: (a list of items which are an existing file name)
         volume fractions of isotropic compartments
in_frac: (a list of items which are an existing file name)
         volume fraction of each fiber

[Optional]
in_bval: (an existing file name)
         input bvals file
in_bvec: (an existing file name)
         input bvecs file
out_bvec: (a file name, nipy default value: bvec.sim)
         simulated b vectors
out_bval: (a file name, nipy default value: bval.sim)
         simulated b values
num_dirs: (an integer (int or long), nipy default value: 32)
         number of gradient directions (when table is automatically
         generated)
in_mask: (an existing file name)
         mask to simulate data
diff_iso: (a list of items which are a float, nipy default value:
         [0.003, 0.00096, 0.00068])
         Diffusivity of isotropic compartments
snr: (an integer (int or long), nipy default value: 0)
         signal-to-noise ratio (dB)
gradients: (an existing file name)
         gradients file
out_mask: (a file name, nipy default value: sim_msk.nii.gz)
         file with the mask simulated
n_proc: (an integer (int or long), nipy default value: 0)
```

(continues on next page)

(continued from previous page)

```

        number of processes
bvalues: (a list of items which are an integer (int or long), nipy
        default value: [1000, 3000])
        list of b-values (when table is automatically generated)
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
out_file: (a file name, nipy default value: sim_dwi.nii.gz)
        output file with fractions to be simulated

```

Outputs:

```

out_bval: (an existing file name)
        simulated b values
out_file: (an existing file name)
        simulated DWIs
out_bvec: (an existing file name)
        simulated b vectors
out_mask: (an existing file name)
        mask file

```

59.6 interfaces.dipy.tensors

59.6.1 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]
mask_file: (an existing file name)
        An optional white matter mask
out_prefix: (a unicode string)
        output prefix for file names
b0_thres: (an integer (int or long), nipy default value: 700)
        b0 threshold

```

Outputs:


```

fa_file: (an existing file name)
rd_file: (an existing file name)
out_file: (an existing file name)
ad_file: (an existing file name)
md_file: (an existing file name)
color_fa_file: (an existing file name)

```

59.6.2 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]
mask_file: (an existing file name)
            An optional white matter mask
out_prefix: (a unicode string)
            output prefix for file names
b0_thres: (an integer (int or long), nipy default value: 700)
          b0 threshold

```

Outputs:

```

out_file: (an existing file name)

```

59.7 interfaces.dipy.tracks

59.7.1 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]
num_seeds: (an integer (int or long), nipy default value: 10000)
    desired number of tracks in tractography
min_angle: (a float, nipy default value: 25.0)
    minimum separation angle
multiprocess: (a boolean, nipy default value: True)
    use multiprocessing
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
in_file: (an existing file name)
    input diffusion data
gfa_thresh: (a float, nipy default value: 0.2)
    GFA threshold to compute tracking mask

[Optional]
seed_mask: (an existing file name)
    input mask within which perform seeding
in_peaks: (an existing file name)
    peaks computed from the odf
seed_coord: (an existing file name)
    file containing the list of seed voxel coordinates (N,3)
out_prefix: (a unicode string)
    output prefix for file names
in_model: (an existing file name)
    input f/d-ODF model extracted from.
tracking_mask: (an existing file name)
    input mask within which perform tracking
```

Outputs:

```
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.
gfa: (a file name)
    The resulting GFA (generalized FA) computed using the peaks of the
    ODF
tracks: (a file name)
    TrackVis file containing extracted streamlines
```

59.7.2 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]
out_filename: (a file name, nipy default value: tdi.nii)
              The output filename for the tracks in TrackVis (.trk) format
reference: (an existing file name)
           A reference file to define RAS coordinates space
points_space: ('rasmm' or 'voxel' or None, nipy default value:
              rasmm)
              coordinates of trk file
voxel_dims: (a list of from 3 to 3 items which are a float)
            The size of each voxel in mm.
data_dims: (a list of from 3 to 3 items which are an integer (int or
            long))
            The size of the image in voxels.
```

Outputs:

```
out_file: (an existing file name)
```


60.1 interfaces.dtitk.base

60.1.1 CommandLineDtitk

[Link to code](#)

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
```

Outputs:

```
None
```

60.2 interfaces.dtitk.registration

60.2.1 AffScalarVol

[Link to code](#)

Wraps the executable command affineScalarVolume.

Applies affine transform to a scalar volume

Example

```
>>> from nipy.interfaces import dtitk
>>> node = dtitk.AffScalarVol()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.transform = 'im_affine.aff'
```

(continues on next page)

(continued from previous page)

```
>>> node.cmdline
'affineScalarVolume -in im1.nii -interp 0 -out im1_affxfmd.nii -trans
im_affine.aff'
>>> node.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        moving scalar volume
        argument: ``-in %s``

[Optional]
interpolation: ('trilinear' or 'NN', nipy default value: trilinear)
               trilinear or nearest neighbor interpolation
               argument: ``-interp %s``
target: (an existing file name)
        output volume specification read from the target volume if specified
        argument: ``-target %s``
        mutually_exclusive: transform
euler: (a tuple of the form: (a float, a float, a float))
       (theta, phi, psi) in degrees
       argument: ``-euler %g %g %g``
       mutually_exclusive: transform
out_file: (a file name)
         output filename
         argument: ``-out %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
translation: (a tuple of the form: (a float, a float, a float))
            translation (x,y,z) in mm
            argument: ``-translation %g %g %g``
            mutually_exclusive: transform
transform: (an existing file name)
          transform to apply: specify an input transformation file; parameters
          input will be ignored
          argument: ``-trans %s``
          mutually_exclusive: target, translation, euler, deformation
deformation: (a tuple of the form: (a float, a float, a float, a
          float, a float, a float))
            (xx,yy,zz,xy,yz,xz)
            argument: ``-deformation %g %g %g %g %g %g``
            mutually_exclusive: transform
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
out_file: (an existing file name)
         moved volume
```

60.2.2 AffSymTensor3DVol[Link to code](#)Wraps the executable command `affineSymTensor3DVolume`.

Applies affine transform to a tensor volume

Example

```
>>> from nipy.interfaces import dtitk
>>> node = dtitk.AffSymTensor3DVOL()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.transform = 'im_affine.aff'
>>> node.cmdline
'affineSymTensor3DVolume -in im1.nii -interp LEI -out im1_affxfmd.nii
-reorient PPD -trans im_affine.aff'
>>> node.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         moving tensor volume
         argument: ``-in %s``

[Optional]
interpolation: ('LEI' or 'EI', nipy default value: LEI)
               Log Euclidean/Euclidean Interpolation
               argument: ``-interp %s``
reorient: ('PPD' or 'NO' or 'FS', nipy default value: PPD)
           Reorientation strategy: preservation of principal direction, no
           reorientation, or finite strain
           argument: ``-reorient %s``
euler: (a tuple of the form: (a float, a float, a float))
        (theta, phi, psi) in degrees
        argument: ``-euler %g %g %g``
        mutually_exclusive: transform
out_file: (a file name)
          output filename
          argument: ``-out %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
translation: (a tuple of the form: (a float, a float, a float))
             translation (x,y,z) in mm
             argument: ``-translation %g %g %g``
             mutually_exclusive: transform
transform: (an existing file name)
           transform to apply: specify an input transformation file; parameters
           input will be ignored
           argument: ``-trans %s``
           mutually_exclusive: target, translation, euler, deformation
deformation: (a tuple of the form: (a float, a float, a float, a
                                     float, a float, a float))
             (xx,yy,zz,xy,yz,xz)
             argument: ``-deformation %g %g %g %g %g %g``
             mutually_exclusive: transform
target: (an existing file name)
        output volume specification read from the target volume if specified
        argument: ``-target %s``
        mutually_exclusive: transform
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
```

(continues on next page)

(continued from previous page)

```
value of class 'str', nipy default value: {}
Environment variables
```

Outputs:

```
out_file: (an existing file name)
```

60.2.3 Affine[Link to code](#)Wraps the executable command `dti_affine_reg`.

Performs affine registration between two tensor volumes

Example

```
>>> from nipy.interfaces import dtitk
>>> node = dtitk.Affine()
>>> node.inputs.fixed_file = 'im1.nii'
>>> node.inputs.moving_file = 'im2.nii'
>>> node.inputs.similarity_metric = 'EDS'
>>> node.inputs.sampling_xyz = (4,4,4)
>>> node.inputs.ftol = 0.01
>>> node.inputs.initialize_xfm = 'im_affine.aff'
>>> node.cmdline
'dti_affine_reg im1.nii im2.nii EDS 4 4 4 0.01 im_affine.aff'
>>> node.run()
```

Inputs:

```
[Mandatory]
moving_file: (an existing file name)
    moving tensor volume
    argument: ``%s``, position: 1
fixed_file: (an existing file name)
    fixed tensor volume
    argument: ``%s``, position: 0
similarity_metric: ('EDS' or 'GDS' or 'DDS' or 'NMI', nipy default
    value: EDS)
    similarity metric
    argument: ``%s``, position: 2
sampling_xyz: (a tuple of the form: (a value of class 'int', a value
    of class 'int', a value of class 'int'), nipy default value: (4,
    4, 4))
    dist between samp points (mm) (x,y,z)
    argument: ``%g %g %g``, position: 3
ftol: (a float, nipy default value: 0.01)
    cost function tolerance
    argument: ``%g``, position: 4

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
initialize_xfm: (an existing file name)
    Initialize w/DTITK-FORMATAffine
    argument: ``%s``, position: 5
```

(continues on next page)

(continued from previous page)

```
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
Environment variables
```

Outputs:

```
out_file: (an existing file name)
out_file_xfm: (an existing file name)
```

60.2.4 AffineTask[Link to code](#)

Wraps the executable command dti_affine_reg.

Inputs:

```
[Mandatory]
moving_file: (an existing file name)
              moving tensor volume
              argument: ``%s``, position: 1
fixed_file: (an existing file name)
              fixed tensor volume
              argument: ``%s``, position: 0
similarity_metric: ('EDS' or 'GDS' or 'DDS' or 'NMI', nipy default
                    value: EDS)
                    similarity metric
                    argument: ``%s``, position: 2
sampling_xyz: (a tuple of the form: (a value of class 'int', a value
                                     of class 'int', a value of class 'int'), nipy default value: (4,
                                     4, 4))
               dist between samp points (mm) (x,y,z)
               argument: ``%g %g %g``, position: 3
ftol: (a float, nipy default value: 0.01)
       cost function tolerance
       argument: ``%g``, position: 4

[Optional]
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
initialize_xfm: (an existing file name)
                Initialize w/DTITK-FORMATAffine
                argument: ``%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
```

Outputs:

```
out_file: (an existing file name)
out_file_xfm: (an existing file name)
```

60.2.5 ComposeXfm[Link to code](#)

Wraps the executable command dfRightComposeAffine.

Combines diffeomorphic and affine transforms

Example

```
>>> from nipy.interfaces import dtitk
>>> node = dtitk.ComposeXfm()
>>> node.inputs.in_df = 'im_warp.df.nii'
>>> node.inputs.in_aff= 'im_affine.aff'
>>> node.cmdline
'dfRightComposeAffine -aff im_affine.aff -df im_warp.df.nii -out
im_warp_affdf.df.nii'
>>> node.run()
```

Inputs:

```
[Mandatory]
in_aff: (an existing file name)
        affine transform file
        argument: ``-aff %s``
in_df: (an existing file name)
        diffeomorphic warp file
        argument: ``-df %s``

[Optional]
out_file: (a file name)
          output path
          argument: ``-out %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
```

Outputs:

```
out_file: (an existing file name)
```

60.2.6 ComposeXfmTask

[Link to code](#)

Wraps the executable command dfRightComposeAffine.

Inputs:

```
[Mandatory]
in_aff: (an existing file name)
        affine transform file
        argument: ``-aff %s``
in_df: (an existing file name)
        diffeomorphic warp file
        argument: ``-df %s``

[Optional]
out_file: (a file name)
          output path
          argument: ``-out %s``
```

(continues on next page)

(continued from previous page)

```
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
out_file: (an existing file name)
```

60.2.7 Diffeo[Link to code](#)

Wraps the executable command `dti_diffeomorphic_reg`.
 Performs diffeomorphic registration between two tensor volumes

Example

```
>>> from nipy.interfaces import dtitk
>>> node = dtitk.Diffeo()
>>> node.inputs.fixed_file = 'im1.nii'
>>> node.inputs.moving_file = 'im2.nii'
>>> node.inputs.mask_file = 'mask.nii'
>>> node.inputs.legacy = 1
>>> node.inputs.n_iters = 6
>>> node.inputs.ftol = 0.002
>>> node.cmdline
'dti_diffeomorphic_reg im1.nii im2.nii mask.nii 1 6 0.002'
>>> node.run()
```

Inputs:

```
[Mandatory]
legacy: (1, nipy default value: 1)
    legacy parameter; always set to 1
    argument: ``%d``, position: 3
n_iters: (an integer (int or long), nipy default value: 6)
    number of iterations
    argument: ``%d``, position: 4
ftol: (a float, nipy default value: 0.002)
    iteration for the optimization to stop
    argument: ``%g``, position: 5

[Optional]
fixed_file: (an existing file name)
    fixed tensor volume
    argument: ``%s``, position: 0
moving_file: (an existing file name)
    moving tensor volume
    argument: ``%s``, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
mask_file: (an existing file name)
```

(continues on next page)

(continued from previous page)

```

mask
argument: ``%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

```

Outputs:

```

out_file: (an existing file name)
out_file_xfm: (an existing file name)

```

60.2.8 DiffeoScalarVol[Link to code](#)

Wraps the executable command deformationScalarVolume.

Applies diffeomorphic transform to a scalar volume

Example

```

>>> from nipy.interfaces import dtitk
>>> node = dtitk.DiffeoScalarVol()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.transform = 'im_warp.df.nii'
>>> node.cmdline
'deformationScalarVolume -in im1.nii -interp 0 -out im1_diffeoxfmd.nii
-trans im_warp.df.nii'
>>> node.run()

```

Inputs:

```

[Mandatory]
transform: (an existing file name)
           transform to apply
           argument: ``-trans %s``
in_file: (an existing file name)
          moving scalar volume
          argument: ``-in %s``

[Optional]
interpolation: ('trilinear' or 'NN', nipy default value: trilinear)
               trilinear, or nearest neighbor
               argument: ``-interp %s``
target: (an existing file name)
         output volume specification read from the target volume if specified
         argument: ``-target %s``
         mutually_exclusive: voxel_size
out_file: (a file name)
          output filename
          argument: ``-out %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
flip: (a tuple of the form: (an integer (int or long), an integer
                             (int or long), an integer (int or long)))
      argument: ``-flip %d %d %d``

```

(continues on next page)

(continued from previous page)

```

resampling_type: ('backward' or 'forward')
    use backward or forward resampling
    argument: ``-type %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
    xyz voxel size (superseded by target)
    argument: ``-vsize %g %g %g``
    mutually_exclusive: target

```

Outputs:

```

out_file: (an existing file name)
    moved volume

```

60.2.9 DiffeoSymTensor3DVOL[Link to code](#)

Wraps the executable command deformationSymTensor3DVolume.

Applies diffeomorphic transform to a tensor volume

Example

```

>>> from nipy.interfaces import dtitk
>>> node = dtitk.DiffeoSymTensor3DVOL()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.transform = 'im_warp.df.nii'
>>> node.cmdline
'deformationSymTensor3DVolume -df FD -in im1.nii -interp LEI -out
im1_diffeoxfmd.nii -reorient PPD -trans im_warp.df.nii'
>>> node.run()

```

Inputs:

```

[Mandatory]
transform: (an existing file name)
    transform to apply
    argument: ``-trans %s``
in_file: (an existing file name)
    moving tensor volume
    argument: ``-in %s``

[Optional]
interpolation: ('LEI' or 'EI', nipy default value: LEI)
    Log Euclidean/Euclidean Interpolation
    argument: ``-interp %s``
out_file: (a file name)
    output filename
    argument: ``-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
df: (a unicode string, nipy default value: FD)

```

(continues on next page)

(continued from previous page)

```

    argument: ``-df %s``
resampling_type: ('backward' or 'forward')
    use backward or forward resampling
    argument: ``-type %s``
voxel_size: (a tuple of the form: (a float, a float, a float))
    xyz voxel size (superseded by target)
    argument: ``-vsize %g %g %g``
mutually_exclusive: target
reorient: ('PPD' or 'FS', nipy default value: PPD)
    Reorientation strategy: preservation of principal direction or
    finite strain
    argument: ``-reorient %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
flip: (a tuple of the form: (an integer (int or long), an integer
    (int or long), an integer (int or long)))
    argument: ``-flip %d %d %d``
target: (an existing file name)
    output volume specification read from the target volume if specified
    argument: ``-target %s``
mutually_exclusive: voxel_size

```

Outputs:

```

out_file: (an existing file name)

```

60.2.10 DiffeoTask[Link to code](#)Wraps the executable command `dti_diffeomorphic_reg`.**Inputs:**

```

[Mandatory]
legacy: (1, nipy default value: 1)
    legacy parameter; always set to 1
    argument: ``%d``, position: 3
n_iters: (an integer (int or long), nipy default value: 6)
    number of iterations
    argument: ``%d``, position: 4
ftol: (a float, nipy default value: 0.002)
    iteration for the optimization to stop
    argument: ``%g``, position: 5

[Optional]
fixed_file: (an existing file name)
    fixed tensor volume
    argument: ``%s``, position: 0
moving_file: (an existing file name)
    moving tensor volume
    argument: ``%s``, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
mask_file: (an existing file name)
    mask

```

(continues on next page)

(continued from previous page)

```

argument: ``%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

```

Outputs:

```

out_file: (an existing file name)
out_file_xfm: (an existing file name)

```

60.2.11 Rigid[Link to code](#)Wraps the executable command `dti_rigid_reg`.

Performs rigid registration between two tensor volumes

Example

```

>>> from nipype.interfaces import dtitk
>>> node = dtitk.Rigid()
>>> node.inputs.fixed_file = 'im1.nii'
>>> node.inputs.moving_file = 'im2.nii'
>>> node.inputs.similarity_metric = 'EDS'
>>> node.inputs.sampling_xyz = (4,4,4)
>>> node.inputs.ftol = 0.01
>>> node.cmdline
'dti_rigid_reg im1.nii im2.nii EDS 4 4 4 0.01'
>>> node.run()

```

Inputs:

```

[Mandatory]
moving_file: (an existing file name)
             moving tensor volume
             argument: ``%s``, position: 1
fixed_file: (an existing file name)
             fixed tensor volume
             argument: ``%s``, position: 0
similarity_metric: ('EDS' or 'GDS' or 'DDS' or 'NMI', nipype default
                   value: EDS)
                   similarity metric
                   argument: ``%s``, position: 2
sampling_xyz: (a tuple of the form: (a value of class 'int', a value
                                     of class 'int', a value of class 'int'), nipype default value: (4,
                                     4, 4))
               dist between samp points (mm) (x,y,z)
               argument: ``%g %g %g``, position: 3
ftol: (a float, nipype default value: 0.01)
      cost function tolerance
      argument: ``%g``, position: 4

[Optional]
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``

```

(continues on next page)

(continued from previous page)

```

initialize_xfm: (an existing file name)
    Initialize w/DTITK-FORMATAffine
    argument: ``%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

```

Outputs:

```

out_file: (an existing file name)
out_file_xfm: (an existing file name)

```

60.2.12 RigidTask[Link to code](#)Wraps the executable command `dti_rigid_reg`.**Inputs:**

```

[Mandatory]
moving_file: (an existing file name)
    moving tensor volume
    argument: ``%s``, position: 1
fixed_file: (an existing file name)
    fixed tensor volume
    argument: ``%s``, position: 0
similarity_metric: ('EDS' or 'GDS' or 'DDS' or 'NMI', nipy default
    value: EDS)
    similarity metric
    argument: ``%s``, position: 2
sampling_xyz: (a tuple of the form: (a value of class 'int', a value
    of class 'int', a value of class 'int'), nipy default value: (4,
    4, 4))
    dist between samp points (mm) (x,y,z)
    argument: ``%g %g %g``, position: 3
ftol: (a float, nipy default value: 0.01)
    cost function tolerance
    argument: ``%g``, position: 4

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
initialize_xfm: (an existing file name)
    Initialize w/DTITK-FORMATAffine
    argument: ``%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

```

Outputs:

```

out_file: (an existing file name)
out_file_xfm: (an existing file name)

```


60.2.13 affScalarVolTask

[Link to code](#)

Wraps the executable command affineScalarVolume.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        moving scalar volume
        argument: ``-in %s``

[Optional]
interpolation: ('trilinear' or 'NN', nipy default value: trilinear)
               trilinear or nearest neighbor interpolation
               argument: ``-interp %s``
target: (an existing file name)
        output volume specification read from the target volume if specified
        argument: ``-target %s``
        mutually_exclusive: transform
euler: (a tuple of the form: (a float, a float, a float))
       (theta, phi, psi) in degrees
       argument: ``-euler %g %g %g``
       mutually_exclusive: transform
out_file: (a file name)
         output filename
         argument: ``-out %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
translation: (a tuple of the form: (a float, a float, a float))
            translation (x,y,z) in mm
            argument: ``-translation %g %g %g``
            mutually_exclusive: transform
transform: (an existing file name)
          transform to apply: specify an input transformation file; parameters
          input will be ignored
          argument: ``-trans %s``
          mutually_exclusive: target, translation, euler, deformation
deformation: (a tuple of the form: (a float, a float, a float, a
          float, a float, a float))
            (xx,yy,zz,xy,yz,xz)
            argument: ``-deformation %g %g %g %g %g %g``
            mutually_exclusive: transform
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables

```

Outputs:

```

out_file: (an existing file name)
         moved volume

```

60.2.14 affSymTensor3DVOLTask

[Link to code](#)

Wraps the executable command affineSymTensor3DVolume.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        moving tensor volume
        argument: ``-in %s``

[Optional]
interpolation: ('LEI' or 'EI', nipy default value: LEI)
        Log Euclidean/Euclidean Interpolation
        argument: ``-interp %s``
reorient: ('PPD' or 'NO' or 'FS', nipy default value: PPD)
        Reorientation strategy: preservation of principal direction, no
        reorientation, or finite strain
        argument: ``-reorient %s``
euler: (a tuple of the form: (a float, a float, a float))
        (theta, phi, psi) in degrees
        argument: ``-euler %g %g %g``
        mutually_exclusive: transform
out_file: (a file name)
        output filename
        argument: ``-out %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
translation: (a tuple of the form: (a float, a float, a float))
        translation (x,y,z) in mm
        argument: ``-translation %g %g %g``
        mutually_exclusive: transform
transform: (an existing file name)
        transform to apply: specify an input transformation file; parameters
        input will be ignored
        argument: ``-trans %s``
        mutually_exclusive: target, translation, euler, deformation
deformation: (a tuple of the form: (a float, a float, a float, a
        float, a float, a float))
        (xx,yy,zz,xy,yz,xz)
        argument: ``-deformation %g %g %g %g %g %g``
        mutually_exclusive: transform
target: (an existing file name)
        output volume specification read from the target volume if specified
        argument: ``-target %s``
        mutually_exclusive: transform
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```
out_file: (an existing file name)
```

60.2.15 diffeoScalarVolTask[Link to code](#)

Wraps the executable command deformationScalarVolume.

Inputs:

```
[Mandatory]
```

(continues on next page)

(continued from previous page)

```

transform: (an existing file name)
    transform to apply
    argument: ``-trans %s``
in_file: (an existing file name)
    moving scalar volume
    argument: ``-in %s``

[Optional]
interpolation: ('trilinear' or 'NN', nipy default value: trilinear)
    trilinear, or nearest neighbor
    argument: ``-interp %s``
target: (an existing file name)
    output volume specification read from the target volume if specified
    argument: ``-target %s``
    mutually_exclusive: voxel_size
out_file: (a file name)
    output filename
    argument: ``-out %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
flip: (a tuple of the form: (an integer (int or long), an integer
    (int or long), an integer (int or long)))
    argument: ``-flip %d %d %d``
resampling_type: ('backward' or 'forward')
    use backward or forward resampling
    argument: ``-type %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
    xyz voxel size (superseded by target)
    argument: ``-vsize %g %g %g``
    mutually_exclusive: target

```

Outputs:

```

out_file: (an existing file name)
    moved volume

```

60.2.16 diffeoSymTensor3DVOLTask[Link to code](#)

Wraps the executable command deformationSymTensor3DVolume.

Inputs:

```

[Mandatory]
transform: (an existing file name)
    transform to apply
    argument: ``-trans %s``
in_file: (an existing file name)
    moving tensor volume
    argument: ``-in %s``

[Optional]

```

(continues on next page)

(continued from previous page)

```

interpolation: ('LEI' or 'EI', nipy default value: LEI)
    Log Euclidean/Euclidean Interpolation
    argument: ``-interp %s``
out_file: (a file name)
    output filename
    argument: ``-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
df: (a unicode string, nipy default value: FD)
    argument: ``-df %s``
resampling_type: ('backward' or 'forward')
    use backward or forward resampling
    argument: ``-type %s``
voxel_size: (a tuple of the form: (a float, a float, a float))
    xyz voxel size (superseded by target)
    argument: ``-vsize %g %g %g``
    mutually_exclusive: target
reorient: ('PPD' or 'FS', nipy default value: PPD)
    Reorientation strategy: preservation of principal direction or
    finite strain
    argument: ``-reorient %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
flip: (a tuple of the form: (an integer (int or long), an integer
    (int or long), an integer (int or long)))
    argument: ``-flip %d %d %d``
target: (an existing file name)
    output volume specification read from the target volume if specified
    argument: ``-target %s``
    mutually_exclusive: voxel_size

```

Outputs:

```
out_file: (an existing file name)
```

60.3 interfaces.dtitk.utils

60.3.1 BinThresh

[Link to code](#)

Wraps the executable command BinaryThresholdImageFilter.

Binarizes an image

Example

```

>>> from nipy.interfaces import dtitk
>>> node = dtitk.BinThresh()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.lower_bound = 0
>>> node.inputs.upper_bound = 100
>>> node.inputs.inside_value = 1
>>> node.inputs.outside_value = 0

```

(continues on next page)

(continued from previous page)

```
>>> node.cmdline
'BinaryThresholdImageFilter im1.nii im1_thrbin.nii 0 100 1 0'
>>> node.run()
```

Inputs:

```
[Mandatory]
lower_bound: (a float, nipy default value: 0.01)
    lower bound of binarization range
    argument: ``%g``, position: 2
outside_value: (a float, nipy default value: 0)
    value for voxelsoutside of binarization range
    argument: ``%g``, position: 5
in_file: (an existing file name)
    Image to threshold/binarize
    argument: ``%s``, position: 0
inside_value: (a float, nipy default value: 1)
    value for voxels in binarization range
    argument: ``%g``, position: 4
upper_bound: (a float, nipy default value: 100)
    upper bound of binarization range
    argument: ``%g``, position: 3

[Optional]
out_file: (a file name)
    output path
    argument: ``%s``, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
out_file: (an existing file name)
```

60.3.2 BinThreshTask[Link to code](#)

Wraps the executable command BinaryThresholdImageFilter.

Inputs:

```
[Mandatory]
lower_bound: (a float, nipy default value: 0.01)
    lower bound of binarization range
    argument: ``%g``, position: 2
outside_value: (a float, nipy default value: 0)
    value for voxelsoutside of binarization range
    argument: ``%g``, position: 5
in_file: (an existing file name)
    Image to threshold/binarize
    argument: ``%s``, position: 0
inside_value: (a float, nipy default value: 1)
    value for voxels in binarization range
```

(continues on next page)

(continued from previous page)

```

        argument: ``%g``, position: 4
upper_bound: (a float, nipy default value: 100)
    upper bound of binarization range
        argument: ``%g``, position: 3

[Optional]
out_file: (a file name)
    output path
        argument: ``%s``, position: 1
args: (a unicode string)
    Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```
out_file: (an existing file name)
```

60.3.3 SVAdjustVoxSp[Link to code](#)

Wraps the executable command SVAdjustVoxelspace.

Adjusts the voxel space of a scalar volume

```

>>> from nipy.interfaces import dtitk
>>> node = dtitk.SVAdjustVoxSp()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.target_file = 'im2.nii'
>>> node.cmdline
'SVAdjustVoxelspace -in im1.nii -out im1_avs.nii -target im2.nii'
>>> node.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    scalar volume to modify
        argument: ``-in %s``

[Optional]
target_file: (a file name)
    target volume to match
        argument: ``-target %s``
    mutually_exclusive: voxel_size, origin
out_file: (a file name)
    output path
        argument: ``-out %s``
args: (a unicode string)
    Additional parameters to the command
        argument: ``%s``
origin: (a tuple of the form: (a float, a float, a float))
    xyz origin (superseded by target)
        argument: ``-origin %g %g %g``
    mutually_exclusive: target_file

```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
            xyz voxel size (superseded by target)
            argument: ``-vsize %g %g %g``
            mutually_exclusive: target_file

```

Outputs:

```

out_file: (an existing file name)

```

60.3.4 SVAdjustVoxSpTask

[Link to code](#)

Wraps the executable command SVAdjustVoxelspace.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         scalar volume to modify
         argument: ``-in %s``

[Optional]
target_file: (a file name)
            target volume to match
            argument: ``-target %s``
            mutually_exclusive: voxel_size, origin
out_file: (a file name)
          output path
          argument: ``-out %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
origin: (a tuple of the form: (a float, a float, a float))
        xyz origin (superseded by target)
        argument: ``-origin %g %g %g``
        mutually_exclusive: target_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', nipy default value: {})
         Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
            xyz voxel size (superseded by target)
            argument: ``-vsize %g %g %g``
            mutually_exclusive: target_file

```

Outputs:

```

out_file: (an existing file name)

```

60.3.5 SVResample

[Link to code](#)

Wraps the executable command SVResample.

Resamples a scalar volume

```
>>> from nipy.interfaces import dtitk
>>> node = dtitk.SVResample()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.target_file = 'im2.nii'
>>> node.cmdline
'SVResample -in im1.nii -out im1_resampled.nii -target im2.nii'
>>> node.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to resample
        argument: ``-in %s``

[Optional]
align: ('center' or 'origin')
        how to align output volume to input volume
        argument: ``-align %s``
target_file: (a file name)
             specs read from the target volume
             argument: ``-target %s``
             mutually_exclusive: array_size, voxel_size, origin
out_file: (a file name)
          output path
          argument: ``-out %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
array_size: (a tuple of the form: (an integer (int or long), an
            integer (int or long), an integer (int or long)))
            resampled array size
            argument: ``-size %d %d %d``
            mutually_exclusive: target_file
origin: (a tuple of the form: (a float, a float, a float))
        xyz origin
        argument: ``-origin %g %g %g``
        mutually_exclusive: target_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', nipy default value: {})
         Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
            resampled voxel size
            argument: ``-vsize %g %g %g``
            mutually_exclusive: target_file
```

Outputs:

```
out_file: (an existing file name)
```

60.3.6 SVResampleTask

[Link to code](#)

Wraps the executable command SVResample.

Inputs:


```

[Mandatory]
in_file: (an existing file name)
        image to resample
        argument: ``-in %s``

[Optional]
align: ('center' or 'origin')
        how to align output volume to input volume
        argument: ``-align %s``
target_file: (a file name)
        specs read from the target volume
        argument: ``-target %s``
        mutually_exclusive: array_size, voxel_size, origin
out_file: (a file name)
        output path
        argument: ``-out %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
array_size: (a tuple of the form: (an integer (int or long), an
        integer (int or long), an integer (int or long)))
        resampled array size
        argument: ``-size %d %d %d``
        mutually_exclusive: target_file
origin: (a tuple of the form: (a float, a float, a float))
        xyz origin
        argument: ``-origin %g %g %g``
        mutually_exclusive: target_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', nipyte default value: {})
        Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
        resampled voxel size
        argument: ``-vsize %g %g %g``
        mutually_exclusive: target_file

```

Outputs:

```
out_file: (an existing file name)
```

60.3.7 TVAdjustOriginTask

[Link to code](#)**Wraps the executable command TVAdjustVoxelspace.****Inputs:**

```

[Mandatory]
in_file: (an existing file name)
        tensor volume to modify
        argument: ``-in %s``

[Optional]
target_file: (a file name)
        target volume to match
        argument: ``-target %s``
        mutually_exclusive: voxel_size, origin

```

(continues on next page)

(continued from previous page)

```

out_file: (a file name)
    output path
    argument: ``-out %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
origin: (a tuple of the form: (a float, a float, a float))
    xyz origin (superseded by target)
    argument: ``-origin %g %g %g``
    mutually_exclusive: target_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', nipype default value: {})
    Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
    xyz voxel size (superseded by target)
    argument: ``-vsize %g %g %g``
    mutually_exclusive: target_file

```

Outputs:

```
out_file: (an existing file name)
```

60.3.8 TVAdjustVoxSp[Link to code](#)

Wraps the executable command TVAdjustVoxelspace.

Adjusts the voxel space of a tensor volume

Example

```

>>> from nipype.interfaces import dtitk
>>> node = dtitk.TVAdjustVoxSp()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.target_file = 'im2.nii'
>>> node.cmdline
'TVAdjustVoxelspace -in im1.nii -out im1_avs.nii -target im2.nii'
>>> node.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    tensor volume to modify
    argument: ``-in %s``

[Optional]
target_file: (a file name)
    target volume to match
    argument: ``-target %s``
    mutually_exclusive: voxel_size, origin
out_file: (a file name)
    output path
    argument: ``-out %s``
args: (a unicode string)
    Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
origin: (a tuple of the form: (a float, a float, a float))
        xyz origin (superseded by target)
        argument: ``-origin %g %g %g``
        mutually_exclusive: target_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', nipy default value: {})
        Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
        xyz voxel size (superseded by target)
        argument: ``-vsize %g %g %g``
        mutually_exclusive: target_file

```

Outputs:

```
out_file: (an existing file name)
```

60.3.9 TVAdjustVoxSpTask[Link to code](#)

Wraps the executable command TVAdjustVoxelspace.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        tensor volume to modify
        argument: ``-in %s``

[Optional]
target_file: (a file name)
        target volume to match
        argument: ``-target %s``
        mutually_exclusive: voxel_size, origin
out_file: (a file name)
        output path
        argument: ``-out %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
origin: (a tuple of the form: (a float, a float, a float))
        xyz origin (superseded by target)
        argument: ``-origin %g %g %g``
        mutually_exclusive: target_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', nipy default value: {})
        Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
        xyz voxel size (superseded by target)
        argument: ``-vsize %g %g %g``
        mutually_exclusive: target_file

```

Outputs:

```
out_file: (an existing file name)
```

60.3.10 TVResample

[Link to code](#)

Wraps the executable command TVResample.

Resamples a tensor volume

```

>>> from nipy.interfaces import dtitk
>>> node = dtitk.TVResample()
>>> node.inputs.in_file = 'im1.nii'
>>> node.inputs.target_file = 'im2.nii'
>>> node.cmdline
'TVResample -in im1.nii -out im1_resampled.nii -target im2.nii'
>>> node.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        tensor volume to resample
        argument: ``-in %s``

[Optional]
interpolation: ('LEI' or 'EI')
               Log Euclidean Euclidean Interpolation
               argument: ``-interp %s``
origin: (a tuple of the form: (a float, a float, a float))
        xyz origin
        argument: ``-origin %g %g %g``
        mutually_exclusive: target_file
target_file: (a file name)
             specs read from the target volume
             argument: ``-target %s``
             mutually_exclusive: array_size, voxel_size, origin
out_file: (a file name)
          output path
          argument: ``-out %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
array_size: (a tuple of the form: (an integer (int or long), an
            integer (int or long), an integer (int or long)))
            resampled array size
            argument: ``-size %d %d %d``
            mutually_exclusive: target_file
align: ('center' or 'origin')
       how to align output volume to input volume
       argument: ``-align %s``
environ: (a dictionary with keys which are a 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
voxel_size: (a tuple of the form: (a float, a float, a float))
            resampled voxel size
            argument: ``-vsize %g %g %g``
            mutually_exclusive: target_file

```

Outputs:

```

out_file: (an existing file name)

```

60.3.11 TVResampleTask

[Link to code](#)

Wraps the executable command TVResample.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        tensor volume to resample
        argument: ``-in %s``

[Optional]
interpolation: ('LEI' or 'EI')
               Log Euclidean Euclidean Interpolation
               argument: ``-interp %s``
origin: (a tuple of the form: (a float, a float, a float))
        xyz origin
        argument: ``-origin %g %g %g``
        mutually_exclusive: target_file
target_file: (a file name)
             specs read from the target volume
             argument: ``-target %s``
             mutually_exclusive: array_size, voxel_size, origin
out_file: (a file name)
          output path
          argument: ``-out %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
array_size: (a tuple of the form: (an integer (int or long), an
        integer (int or long), an integer (int or long)))
        resampled array size
        argument: ``-size %d %d %d``
        mutually_exclusive: target_file
align: ('center' or 'origin')
        how to align output volume to input volume
        argument: ``-align %s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
voxel_size: (a tuple of the form: (a float, a float, a float))
            resampled voxel size
            argument: ``-vsize %g %g %g``
            mutually_exclusive: target_file

```

Outputs:

```

out_file: (an existing file name)

```

60.3.12 TVtool

[Link to code](#)

Wraps the executable command TVtool.

Calculates a tensor metric volume from a tensor volume

```

>>> from nipy.interfaces import dtitk
>>> node = dtitk.TVtool()

```

(continues on next page)

(continued from previous page)

```

>>> node.inputs.in_file = 'iml.nii'
>>> node.inputs.in_flag = 'fa'
>>> node.cmdline
'TVtool -in iml.nii -fa -out iml_fa.nii'
>>> node.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         scalar volume to resample
         argument: ``-in %s``

[Optional]
out_file: (a file name)
         argument: ``-out %s``
in_flag: ('fa' or 'tr' or 'ad' or 'rd' or 'pd' or 'rgb')
         argument: ``-%s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables

```

Outputs:

```
out_file: (a file name)
```

60.3.13 TVtoolTask

[Link to code](#)

Wraps the executable command TVtool.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         scalar volume to resample
         argument: ``-in %s``

[Optional]
out_file: (a file name)
         argument: ``-out %s``
in_flag: ('fa' or 'tr' or 'ad' or 'rd' or 'pd' or 'rgb')
         argument: ``-%s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables

```

Outputs:

```
out_file: (a file name)
```

61.1 interfaces.elastix.registration

61.1.1 AnalyzeWarp

[Link to code](#)

Wraps the executable command `transformix`.

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 -threads 1 -out ./ -tp_
↪TransformParameters.0.txt'
```

Inputs:

```
[Mandatory]
output_path: (an existing directory name, nipy default value: ./)
    output directory
    argument: ``-out %s``
transform_file: (an existing file name)
    transform-parameter file, only 1
    argument: ``-tp %s``

[Optional]
jacmat: ('all', nipy default value: all)
    generate an image with the spatial Jacobian matrix at each voxel
    argument: ``-jacmat %s``
points: ('all', nipy default value: all)
    transform all points from the input-image, which effectively
    generates a deformation field.
    argument: ``-def %s``, position: 0
```

(continues on next page)

(continued from previous page)

```

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
jac: ('all', nipy default value: all)
    generate an image with the determinant of the spatial Jacobian
    argument: ``-jac %s``
num_threads: (an integer (int or long), nipy default value: 1)
    set the maximum number of threads of elastix
    argument: ``-threads %01d``
moving_image: (an existing file name)
    input image to deform (not used)
    argument: ``-in %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

disp_field: (a file name)
    displacements field
jacdet_map: (a file name)
    det(Jacobian) map
jacmat_map: (a file name)
    Jacobian matrix map

```

61.1.2 ApplyWarp[Link to code](#)

Wraps the executable 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 -threads 1 -out ./ -tp TransformParameters.0.txt'

```

Inputs:

```

[Mandatory]
output_path: (an existing directory name, nipy default value: ./)
    output directory
    argument: ``-out %s``
transform_file: (an existing file name)
    transform-parameter file, only 1
    argument: ``-tp %s``
moving_image: (an existing file name)
    input image to deform
    argument: ``-in %s``

[Optional]

```

(continues on next page)

(continued from previous page)

```

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long), nipyype default value: 1)
    set the maximum number of threads of elastix
    argument: ``-threads %01d``
environ: (a dictionary with keys which are a 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

```

Outputs:

```

warped_file: (a file name)
    input moving image warped to fixed image

```

61.1.3 PointsWarp

[Link to code](#)

Wraps the executable command transformix.

Use transformix to apply a transform on an input point set. The transform is specified in the transform-parameter file.

Example

```

>>> from nipyype.interfaces.elastix import PointsWarp
>>> reg = PointsWarp()
>>> reg.inputs.points_file = 'surfl.vtk'
>>> reg.inputs.transform_file = 'TransformParameters.0.txt'
>>> reg.cmdline
'transformix -threads 1 -out ./ -def surfl.vtk -tp TransformParameters.0.txt'

```

Inputs:

```

[Mandatory]
output_path: (an existing directory name, nipyype default value: ./)
    output directory
    argument: ``-out %s``
points_file: (an existing file name)
    input points (accepts .vtk triangular meshes).
    argument: ``-def %s``
transform_file: (an existing file name)
    transform-parameter file, only 1
    argument: ``-tp %s``

[Optional]
num_threads: (an integer (int or long), nipyype default value: 1)
    set the maximum number of threads of elastix
    argument: ``-threads %01d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a 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

```

Outputs:

```
warped_file: (a file name)
    input points displaced in fixed image domain
```

61.1.4 Registration

[Link to code](#)

Wraps the executable command `elastix`.

Elastix nonlinear registration interface

Example

```
>>> from nipy.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 -threads 1 -out ./ -p elastix.txt'
```

Inputs:

```
[Mandatory]
moving_image: (an existing file name)
    moving image
    argument: ``-m %s``
parameters: (a list of items which are an existing file name)
    parameter file, elastix handles 1 or more -p
    argument: ``-p %s...``
fixed_image: (an existing file name)
    fixed image
    argument: ``-f %s``
output_path: (an existing directory name, nipy default value: ./)
    output directory
    argument: ``-out %s``

[Optional]
fixed_mask: (an existing file name)
    mask for fixed image
    argument: ``-fMask %s``
initial_transform: (an existing file name)
    parameter file for initial transform
    argument: ``-t0 %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
moving_mask: (an existing file name)
    mask for moving image
    argument: ``-mMask %s``
num_threads: (an integer (int or long), nipy default value: 1)
    set the maximum number of threads of elastix
    argument: ``-threads %01d``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

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

```

61.2 interfaces.elastix.utils

61.2.1 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]
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
              argument: ``ResultImagePixelType``
interpolation: ('cubic' or 'linear' or 'nearest', nipy default
                value: cubic)
                set a new interpolator for transformation
                argument: ``FinalBSplineInterpolationOrder``
output_format: ('nii.gz' or 'nii' or 'mhd' or 'hdr' or 'vtk')
                set a new image format for resampled images
                argument: ``ResultImageFormat``
output_file: (a file name)
              the filename for the resulting transform file
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

```


62.1 interfaces.freesurfer.longitudinal

62.1.1 FuseSegmentations

[Link to code](#)

Wraps the executable 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]
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
    argument: ``%s``, position: -2
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
    argument: ``-n %s``
in_segmentations_noCC: (a list of items which are an existing file
```

(continues on next page)

(continued from previous page)

```

        name)
        name of aseg file w/o CC labels (default: aseg.auto_noCCseg.mgz)
        must include the corresponding file for all the given timepoints
        argument: ``-c %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
        argument: ``-a %s``

[Optional]
subject_id: (a string)
        subject_id being processed
        argument: ``%s``, position: -3
subjects_dir: (an existing directory name)
        subjects directory
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

out_file: (a file name)
        output fused segmentation file

```

62.1.2 RobustTemplate[Link to code](#)

Wraps the executable command `mri_robust_template`.
construct an unbiased robust template for longitudinal volumes

Examples

```

>>> from nipy.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 #doctest:
'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 #doctest:
'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']

```

(continues on next page)

(continued from previous page)

```
>>> 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.inputs.transform_outputs = True
>>> template.inputs.scaled_intensity_outputs = True
>>> template.cmdline
'mri_robust_template --satit --average 0 --fixtp --mov structural.nii functional.
↪nii --inittp 1 --noit --template T1.nii --iscaleout ../is1.txt ../is2.txt --
↪subsample 200 --lta ../tp1.lta ../tp2.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)
    argument: ``--satit``
    mutually_exclusive: outlier_sensitivity
outlier_sensitivity: (a float)
    set outlier sensitivity manually (e.g. "--sat 4.685" ). Higher
    values mean less sensitivity.
    argument: ``--sat %.4f``
    mutually_exclusive: auto_detect_sensitivity
in_files: (a list of items which are an existing file name)
    input movable volumes to be aligned to common mean/median template
    argument: ``--mov %s``
out_file: (a file name, nipy default value:
    mri_robust_template_out.mgz)
    output template volume (final mean/median image)
    argument: ``--template %s``

[Optional]
scaled_intensity_outputs: (a list of items which are a file name or a
    boolean)
    final intensity scales (will activate --iscale)
    argument: ``--iscaleout %s``
initial_timepoint: (an integer (int or long))
    use TP# for spacial init (default random), 0: no init
    argument: ``--inittp %d``
subsample_threshold: (an integer (int or long))
    subsample if dim > # on all axes (default no subs.)
    argument: ``--subsample %d``
subjects_dir: (an existing directory name)
    subjects directory
intensity_scaling: (a boolean)
```

(continues on next page)

(continued from previous page)

```

        allow also intensity scaling (default off)
        argument: ``--iscale``
num_threads: (an integer (int or long))
        allows for specifying more threads
no_iteration: (a boolean)
        do not iterate, just create first template
        argument: ``--noit``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
transform_outputs: (a list of items which are a file name or a
        boolean)
        output xforms to template (for each input)
        argument: ``--lta %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
average_metric: ('median' or 'mean')
        construct template from: 0 Mean, 1 Median (default)
        argument: ``--average %d``
initial_transforms: (a list of items which are an existing file name)
        use initial transforms (lta) on source
        argument: ``--ixforms %s``
in_intensity_scales: (a list of items which are an existing file
        name)
        use initial intensity scales
        argument: ``--iscalein %s``
fixed_timepoint: (a boolean)
        map everthing to init TP# (init TP is not resampled)
        argument: ``--fixtp``

```

Outputs:

```

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
out_file: (an existing file name)
        output template volume (final mean/median image)

```

62.2 interfaces.freesurfer.model

62.2.1 Binarize

[Link to code](#)Wraps the executable 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
        argument: ``--i %s``

[Optional]
merge_file: (an existing file name)
            merge with mergevol
            argument: ``--merge %s``
bin_val_not: (an integer (int or long))
            set vox outside range to val (default is 0)
            argument: ``--binvalnot %d``
count_file: (a boolean or a file name)
            save number of hits in ascii file (hits, ntotvox, pct)
            argument: ``--count %s``
ventricles: (a boolean)
            set match vals those for aseg ventricles+choroid (not 4th)
            argument: ``--ventricles``
environ: (a dictionary with keys which are a 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
max: (a float)
    max thresh
    argument: ``--max %f``
    mutually_exclusive: wm_ven_csf
bin_val: (an integer (int or long))
    set vox within thresh to val (default is 1)
    argument: ``--binval %d``
frame_no: (an integer (int or long))
    use 0-based frame of input (default is 0)
    argument: ``--frame %s``
zero_edges: (a boolean)
    zero the edge voxels
    argument: ``--zero-edges``
mask_thresh: (a float)
    set thresh for mask
    argument: ``--mask-thresh %f``
abs: (a boolean)
    take abs of invol first (ie, make unsigned)
    argument: ``--abs``
erode: (an integer (int or long))
    nerode: erode binarization in 3D (after any dilation)
    argument: ``--erode %d``
invert: (a boolean)
    set binval=0, binvalnot=1
    argument: ``--inv``
rmin: (a float)
    compute min based on rmin*globalmean
    argument: ``--rmin %f``
subjects_dir: (an existing directory name)
    subjects directory
rmax: (a float)
    compute max based on rmax*globalmean
    argument: ``--rmax %f``
erode2d: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

nerode2d: erode binarization in 2D (after any 3D erosion)
argument: ``--erode2d %d``
out_type: ('nii' or 'nii.gz' or 'mgz')
output file type
wm: (a boolean)
set match vals to 2 and 41 (aseg for cerebral WM)
argument: ``--wm``
match: (a list of items which are an integer (int or long))
match instead of threshold
argument: ``--match %d...``
mask_file: (an existing file name)
must be within mask
argument: ``--mask maskvol``
min: (a float)
min thresh
argument: ``--min %f``
mutually_exclusive: wm_ven_csf
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
binary_file: (a file name)
binary output volume
argument: ``--o %s``
wm_ven_csf: (a boolean)
WM and ventricular CSF, including choroid (not 4th)
argument: ``--wm+vcsf``
mutually_exclusive: min, max
dilate: (an integer (int or long))
niters: dilate binarization in 3D
argument: ``--dilate %d``
zero_slice_edge: (a boolean)
zero the edge slice voxels
argument: ``--zero-slice-edges``
bin_col_num: (a boolean)
set binarized voxel value to its column number
argument: ``--bincol``

```

Outputs:

```

binary_file: (an existing file name)
binarized output volume
count_file: (a file name)
ascii file containing number of hits

```

62.2.2 Concatenate[Link to code](#)

Wraps the executable 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
    argument: ``--i %s...``

[Optional]
combine: (a boolean)
    Combine non-zero values into single frame volume
    argument: ``--combine``
paired_stats: ('sum' or 'avg' or 'diff' or 'diff-norm' or 'diff-
    norm1' or 'diff-norm2')
    Compute paired sum, avg, or diff
    argument: ``--paired-%s``
sort: (a boolean)
    Sort each voxel by ascending frame value
    argument: ``--sort``
subjects_dir: (an existing directory name)
    subjects directory
keep_dtype: (a boolean)
    Keep voxelwise precision type (default is float
    argument: ``--keep-datatype``
max_index: (a boolean)
    Compute the index of max voxel in concatenated volumes
    argument: ``--max-index``
vote: (a boolean)
    Most frequent value at each voxel and fraction of occurrences
    argument: ``--vote``
max_bonfcor: (a boolean)
    Compute max and bonferroni correct (assumes -log10(ps))
    argument: ``--max-bonfcor``
mean_div_n: (a boolean)
    compute mean/nframes (good for var)
    argument: ``--mean-div-n``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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
    argument: ``--%s``
gmean: (an integer (int or long))
    create matrix to average Ng groups, Nper=Ntot/Ng
    argument: ``--gmean %d``
mask_file: (an existing file name)
    Mask input with a volume
    argument: ``--mask %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
multiply_by: (a float)

```

(continues on next page)

(continued from previous page)

```

        Multiply input volume by some amount
        argument: ``--mul %f``
add_val: (a float)
        Add some amount to the input volume
        argument: ``--add %f``
sign: ('abs' or 'pos' or 'neg')
        Take only pos or neg voxles from input, or take abs
        argument: ``--%s``
multiply_matrix_file: (an existing file name)
        Multiply input by an ascii matrix in file
        argument: ``--mtx %s``
concatenated_file: (a file name)
        Output volume
        argument: ``--o %s``

```

Outputs:

```

concatenated_file: (an existing file name)
        Path/name of the output volume

```

62.2.3 GLMFit[Link to code](#)Wraps the executable 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
        argument: ``--y %s``

[Optional]
contrast: (a list of items which are an existing file name)
        contrast file
        argument: ``--C %s...``
invert_mask: (a boolean)
        invert mask
        argument: ``--mask-inv``
cortex: (a boolean)
        use subjects ?h.cortex.label as label
        argument: ``--cortex``
mutually_exclusive: label_file
fixed_fx_dof: (an integer (int or long))
        dof for fixed effects analysis
        argument: ``--ffxdof %d``

```

(continues on next page)

(continued from previous page)

```

        mutually_exclusive: fixed_fx_dof_file
diag: (an integer (int or long))
    Gdiag_no : set diagnositc level
    argument: ``--diag %d``
sim_sign: ('abs' or 'pos' or 'neg')
    abs, pos, or neg
    argument: ``--sim-sign %s``
fixed_fx_var: (an existing file name)
    for fixed effects analysis
    argument: ``--yffxvar %s``
synth: (a boolean)
    replace input with gaussian
    argument: ``--synth``
weighted_ls: (an existing file name)
    weighted least squares
    argument: ``--wls %s``
    mutually_exclusive: weight_file, weight_inv, weight_sqrt
per_voxel_reg: (a list of items which are an existing file name)
    per-voxel regressors
    argument: ``--pvr %s...``
one_sample: (a boolean)
    construct X and C as a one-sample group mean
    argument: ``--osgm``
    mutually_exclusive: one_sample, fsgd, design, contrast
pca: (a boolean)
    perform pca/svd analysis on residual
    argument: ``--pca``
fsgd: (a tuple of the form: (an existing file name, 'doss' or
    'dods'))
    freesurfer descriptor file
    argument: ``--fsgd %s %s``
    mutually_exclusive: fsgd, design, one_sample
diag_cluster: (a boolean)
    save sig volume and exit from first sim loop
    argument: ``--diag-cluster``
weight_file: (an existing file name)
    weight for each input at each voxel
    mutually_exclusive: weighted_ls
sim_done_file: (a file name)
    create file when simulation finished
    argument: ``--sim-done %s``
save_cond: (a boolean)
    flag to save design matrix condition at each voxel
    argument: ``--save-cond``
subject_id: (a unicode string)
    subject id for surface geometry
save_res_corr_mtx: (a boolean)
    save residual error spatial correlation matrix (eres.scm). Big!
    argument: ``--eres-scm``
uniform: (a tuple of the form: (a float, a float))
    use uniform distribution instead of gaussian
    argument: ``--uniform %f %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
calc_AR1: (a boolean)
    compute and save temporal AR1 of residual

```

(continues on next page)

(continued from previous page)

```

        argument: ``--tar1``
compute_log_y: (a boolean)
    compute natural log of y prior to analysis
    argument: ``--logy``
save_residual: (a boolean)
    save residual error (eres)
    argument: ``--eres-save``
debug: (a boolean)
    turn on debugging
    argument: ``--debug``
allow_repeated_subjects: (a boolean)
    allow subject names to repeat in the fsgd file (must appear before
    --fsgd
    argument: ``--allowsubprep``
save_estimate: (a boolean)
    save signal estimate (yhat)
    argument: ``--yhat-save``
resynth_test: (an integer (int or long))
    test GLM by resynthesis
    argument: ``--resynthtest %d``
design: (an existing file name)
    design matrix file
    argument: ``--X %s``
    mutually_exclusive: fsgd, design, one_sample
check_opts: (a boolean)
    don't run anything, just check options and exit
    argument: ``--checkopts``
surf: (a boolean)
    analysis is on a surface mesh
    argument: ``--surf %s %s %s``
    requires: subject_id, hemi
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
prune: (a boolean)
    remove voxels that do not have a non-zero value at each frame (def)
    argument: ``--prune``
weight_inv: (a boolean)
    invert weights
    argument: ``--w-inv``
    mutually_exclusive: weighted_ls
no_mask_smooth: (a boolean)
    do not mask when smoothing
    argument: ``--no-mask-smooth``
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
    argument: ``--voxdump %d %d %d``
surf_geo: (a unicode string, nipy default value: white)
    surface geometry name (e.g. white, pial)
prune_thresh: (a float)
    prune threshold. Default is FLT_MIN
    argument: ``--prune_thr %f``
    mutually_exclusive: noprune
no_prune: (a boolean)
    do not prune

```

(continues on next page)

(continued from previous page)

```

        argument: ``--no-prune``
        mutually_exclusive: prunethresh
weight_sqrt: (a boolean)
    sqrt of weights
    argument: ``--w-sqrt``
    mutually_exclusive: weighted_ls
profile: (an integer (int or long))
    niters : test speed
    argument: ``--profile %d``
no_contrast_ok: (a boolean)
    do not fail if no contrasts specified
    argument: ``--no-contrasts-ok``
allow_ill_cond: (a boolean)
    allow ill-conditioned design matrices
    argument: ``--illcond``
subjects_dir: (an existing directory name)
    subjects directory
label_file: (an existing file name)
    use label as mask, surfaces only
    argument: ``--label %s``
    mutually_exclusive: cortex
glm_dir: (a unicode string)
    save outputs to dir
    argument: ``--glmdir %s``
seed: (an integer (int or long))
    used for synthesizing noise
    argument: ``--seed %d``
fixed_fx_dof_file: (a file name)
    text file with dof for fixed effects analysis
    argument: ``--ffxdofdat %d``
    mutually_exclusive: fixed_fx_dof
hemi: ('lh' or 'rh')
    surface hemisphere
mask_file: (an existing file name)
    binary mask
    argument: ``--mask %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
    argument: ``--sim %s %d %f %s``
no_est_fwhm: (a boolean)
    turn off FWHM output estimation
    argument: ``--no-est-fwhm``
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
    argument: ``--selfreg %d %d %d``
fwhm: (a floating point number >= 0.0)
    smooth input by fwhm
    argument: ``--fwhm %f``
force_perm: (a boolean)
    force permutation test, even when design matrix is not orthog
    argument: ``--perm-force``
var_fwhm: (a floating point number >= 0.0)
    smooth variance by fwhm
    argument: ``--var-fwhm %f``

```

Outputs:

```

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
svd_stats_file: (a file name)
    text file summarizing the residual PCA
gamma_file: (a list of items which are any value)
    map of contrast of regression coefficients
glm_dir: (an existing directory name)
    output directory
singular_values: (a file name)
    matrix singular values from residual PCA
beta_file: (an existing file name)
    map of regression coefficients
fwhm_file: (a file name)
    text file with estimated smoothness
sig_file: (a list of items which are any value)
    map of F-test significance (in -log10p)
mask_file: (a file name)
    map of the mask used in the analysis
dof_file: (a file name)
    text file with effective degrees-of-freedom for the analysis
estimate_file: (a file name)
    map of the estimated Y values
ftest_file: (a list of items which are any value)
    map of test statistic values
gamma_var_file: (a list of items which are any value)
    map of regression contrast variance
spatial_eigenvectors: (a file name)
    map of spatial eigenvectors from residual PCA
frame_eigenvectors: (a file name)
    matrix of frame eigenvectors from residual PCA

```

62.2.4 Label2Annot

[Link to code](#)

Wraps the executable command `mris_label2annot`.

Converts a set of surface labels to an annotation file

Examples

```

>>> from nipy.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]
subject_id: (a string, nipy default value: subject_id)
    Subject name/ID
    argument: ``--s %s``
orig: (an existing file name)
    implicit {hemisphere}.orig
out_annot: (a string)
    Name of the annotation to create
    argument: ``--a %s``
in_labels: (a list of items which are any value)
    List of input label files
    argument: ``--l %s...``
hemisphere: ('lh' or 'rh')
    Input hemisphere
    argument: ``--hemi %s``

[Optional]
subjects_dir: (an existing directory name)
    subjects directory
color_table: (an existing file name)
    File that defines the structure names, their indices, and their
    color
    argument: ``--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', nipy default value: {})
    Environment variables
keep_max: (a boolean)
    Keep label with highest 'stat' value
    argument: ``--maxstatwinner``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose_off: (a boolean)
    Turn off overlap and stat override messages
    argument: ``--noverbose``

```

Outputs:

```

out_file: (an existing file name)
    Output annotation file

```

62.2.5 Label2Label

[Link to code](#)

Wraps the executable 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 nipyne.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]
subject_id: (a string, nipyne default value: subject_id)
    Target subject
    argument: ``--trgsubject %s``
source_sphere_reg: (an existing file name)
    Implicit input <hemisphere>.sphere.reg
source_subject: (a string)
    Source subject name
    argument: ``--srcsubject %s``
white: (an existing file name)
    Implicit input <hemisphere>.white
hemisphere: ('lh' or 'rh')
    Input hemisphere
    argument: ``--hemi %s``
source_white: (an existing file name)
    Implicit input <hemisphere>.white
sphere_reg: (an existing file name)
    Implicit input <hemisphere>.sphere.reg
source_label: (an existing file name)
    Source label
    argument: ``--srclabel %s``

[Optional]
subjects_dir: (an existing directory name)
    subjects directory
registration_method: ('surface' or 'volume', nipyne default value:
    surface)
    Registration method
    argument: ``--regmethod %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', nipyne default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
out_file: (a file name)
```

(continues on next page)

(continued from previous page)

```
Target label
argument: ``--trglabel %s``
```

Outputs:

```
out_file: (an existing file name)
Output label
```

62.2.6 Label2Vol[Link to code](#)Wraps the executable 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 0.5 --label cortex.label --reg register.dat --temp
↳ structural.nii --o foo_out.nii'
```

Inputs:

```
[Mandatory]
seg_file: (an existing file name)
    segmentation file
    argument: ``--seg %s``
    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
    argument: ``--label %s...``
    mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg
annot_file: (an existing file name)
    surface annotation file
    argument: ``--annot %s``
    mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg
    requires: subject_id, hemi
template_file: (an existing file name)
    output template volume
    argument: ``--temp %s``
aparc_aseg: (a boolean)
    use aparc+aseg.mgz in subjectdir as seg
    argument: ``--aparc+aseg``
    mutually_exclusive: label_file, annot_file, seg_file, aparc_aseg

[Optional]
native_vox2ras: (a boolean)
    use native vox2ras xform instead of tkregister-style
    argument: ``--native-vox2ras``
label_voxel_volume: (a float)
    volume of each label point (def 1mm3)
    argument: ``--labvoxvol %f``
proj: (a tuple of the form: ('abs' or 'frac', a float, a float, a
    float))
    project along surface normal
```

(continues on next page)

(continued from previous page)

```

        argument: ``--proj %s %f %f %f``
        requires: subject_id, hemi
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
surface: (a unicode string)
        use surface instead of white
        argument: ``--surf %s``
reg_header: (an existing file name)
        label template volume
        argument: ``--regheader %s``
        mutually_exclusive: reg_file, reg_header, identity
label_hit_file: (a file name)
        file with each frame is nhits for a label
        argument: ``--hits %s``
reg_file: (an existing file name)
        tkregister style matrix VolXYZ = R*LabelXYZ
        argument: ``--reg %s``
        mutually_exclusive: reg_file, reg_header, identity
vol_label_file: (a file name)
        output volume
        argument: ``--o %s``
subjects_dir: (an existing directory name)
        subjects directory
invert_mtx: (a boolean)
        Invert the registration matrix
        argument: ``--invertmtx``
fill_thresh: (0.0 <= a floating point number <= 1.0)
        thresh : between 0 and 1
        argument: ``--fillthresh %g``
map_label_stat: (a file name)
        map the label stats field into the vol
        argument: ``--label-stat %s``
subject_id: (a unicode string)
        subject id
        argument: ``--subject %s``
hemi: ('lh' or 'rh')
        hemisphere to use lh or rh
        argument: ``--hemi %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
identity: (a boolean)
        set R=I
        argument: ``--identity``
        mutually_exclusive: reg_file, reg_header, identity

```

Outputs:

```

vol_label_file: (an existing file name)
        output volume

```

62.2.7 MRISPreproc[Link to code](#)Wraps the executable command `mriss_preproc`.

Use FreeSurfer `mris_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
'mris_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
    argument: ``--hemi %s``
target: (a unicode string)
    target subject name
    argument: ``--target %s``

[Optional]
fwhm: (a float)
    smooth by fwhm mm on the target surface
    argument: ``--fwhm %f``
    mutually_exclusive: num_iters
fwhm_source: (a float)
    smooth by fwhm mm on the source surface
    argument: ``--fwhm-src %f``
    mutually_exclusive: num_iters_source
surf_area: (a unicode string)
    Extract vertex area from subject/surf/hemi.surfname to use as input.
    argument: ``--area %s``
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
subjects_dir: (an existing directory name)
    subjects directory
surf_measure_file: (a list of items which are an existing file name)
    file alternative to surfmeas, still requires list of subjects
    argument: ``--is %s...``
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
subject_file: (an existing file name)
    file specifying subjects separated by white space
    argument: ``--f %s``
    mutually_exclusive: subjects, fsgd_file, subject_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', nipy default value: {})
    Environment variables
num_iters: (an integer (int or long))
    niters : smooth by niters on the target surface
    argument: ``--niters %d``
    mutually_exclusive: fwhm
source_format: (a unicode string)
    source format
    argument: ``--srcfmt %s``
```

(continues on next page)

(continued from previous page)

```
fsgd_file: (an existing file name)
    specify subjects using fsgd file
    argument: ``--fsgd %s``
    mutually_exclusive: subjects, fsgd_file, subject_file
surf_dir: (a unicode string)
    alternative directory (instead of surf)
    argument: ``--surfdir %s``
out_file: (a file name)
    output filename
    argument: ``--out %s``
surf_measure: (a unicode string)
    Use subject/surf/hemi.surf_measure as input
    argument: ``--meas %s``
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_iters_source: (an integer (int or long))
    niters : smooth by niters on the source surface
    argument: ``--niterssrc %d``
    mutually_exclusive: fwhm_source
smooth_cortex_only: (a boolean)
    only smooth cortex (ie, exclude medial wall)
    argument: ``--smooth-cortex-only``
subjects: (a list of items which are any value)
    subjects from who measures are calculated
    argument: ``--s %s...``
    mutually_exclusive: subjects, fsgd_file, subject_file
proj_frac: (a float)
    projection fraction for vol2surf
    argument: ``--projfrac %s``
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
    argument: ``--iv %s %s...``
```

Outputs:

```
out_file: (a file name)
    preprocessed output file
```

62.2.8 MRISPreprocReconAll

[Link to code](#)

Wraps the executable 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'
```

(continues on next page)

(continued from previous page)

```
>>> 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
    argument: ``--hemi %s``
target: (a unicode string)
    target subject name
    argument: ``--target %s``

[Optional]
fwhm_source: (a float)
    smooth by fwhm mm on the source surface
    argument: ``--fwhm-src %f``
    mutually_exclusive: num_iters_source
surf_area: (a unicode string)
    Extract vertex area from subject/surf/hemi.surfname to use as input.
    argument: ``--area %s``
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
surf_measure_file: (an existing file name)
    file necessary for surfmeas
    argument: ``--meas %s``
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
num_iters: (an integer (int or long))
    niters : smooth by niters on the target surface
    argument: ``--niters %d``
    mutually_exclusive: fwhm
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)
    argument: ``--smooth-cortex-only``
surf_measure: (a unicode string)
    Use subject/surf/hemi.surf_measure as input
    argument: ``--meas %s``
    mutually_exclusive: surf_measure, surf_measure_file, surf_area
proj_frac: (a float)
    projection fraction for vol2surf
    argument: ``--projfrac %s``
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
    argument: ``--iv %s %s...``
source_format: (a unicode string)
    source format
    argument: ``--srcfmt %s``
subjects: (a list of items which are any value)
    subjects from who measures are calculated
```

(continues on next page)

(continued from previous page)

```

    argument: ``--s %s...``
    mutually_exclusive: subjects, fsgd_file, subject_file
subjects_dir: (an existing directory name)
    subjects directory
copy_inputs: (a boolean)
    If running as a node, set this to True this will copy some implicit
    inputs to the node directory.
subject_id: (a string, nipyne default value: subject_id)
    subject from whom measures are calculated
    argument: ``--s %s``
    mutually_exclusive: subjects, fsgd_file, subject_file, subject_id
fsgd_file: (an existing file name)
    specify subjects using fsgd file
    argument: ``--fsgd %s``
    mutually_exclusive: subjects, fsgd_file, subject_file
surfreg_files: (a list of items which are an existing file name)
    lh and rh input surface registration files
    argument: ``--surfreg %s``
    requires: lh_surfreg_target, rh_surfreg_target
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_iters_source: (an integer (int or long))
    niters : smooth by niters on the source surface
    argument: ``--niterssrc %d``
    mutually_exclusive: fwhm_source
subject_file: (an existing file name)
    file specifying subjects separated by white space
    argument: ``--f %s``
    mutually_exclusive: subjects, fsgd_file, subject_file
lh_surfreg_target: (a file name)
    Implicit target surface registration file
    requires: surfreg_files
surf_dir: (a unicode string)
    alternative directory (instead of surf)
    argument: ``--surfdir %s``
out_file: (a file name)
    output filename
    argument: ``--out %s``
fwhm: (a float)
    smooth by fwhm mm on the target surface
    argument: ``--fwhm %f``
    mutually_exclusive: num_iters

```

Outputs:

```

out_file: (a file name)
    preprocessed output file

```

62.2.9 MS_LDA[Link to code](#)Wraps the executable 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]
vol_synth_file: (a file name)
    filename for the synthesized output volume
    argument: ``-synth %s``
weight_file: (a file name)
    filename for the LDA weights (input or output)
    argument: ``-weight %s``
images: (a list of items which are an existing file name)
    list of input FLASH images
    argument: ``%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
    argument: ``-lda %s``

[Optional]
shift: (an integer (int or long))
    shift all values equal to the given value to zero
    argument: ``-shift %d``
use_weights: (a boolean)
    Use the weights from a previously generated weight file
    argument: ``-W``
subjects_dir: (an existing directory name)
    subjects 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
label_file: (a file name)
    filename of the label volume
    argument: ``-label %s``
mask_file: (a file name)
    filename of the brain mask volume
    argument: ``-mask %s``
conform: (a boolean)
    Conform the input volumes (brain mask typically already conformed)
    argument: ``-conform``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

```

Outputs:

```
vol_synth_file: (an existing file name)
weight_file: (an existing file name)
```

62.2.10 OneSampleTTest

[Link to code](#)

Wraps the executable command `mri_glmfit`.

Inputs:

```
[Mandatory]
in_file: (a file name)
        input 4D file
        argument: ``--y %s``

[Optional]
contrast: (a list of items which are an existing file name)
        contrast file
        argument: ``--C %s...``
invert_mask: (a boolean)
        invert mask
        argument: ``--mask-inv``
cortex: (a boolean)
        use subjects ?h.cortex.label as label
        argument: ``--cortex``
        mutually_exclusive: label_file
fixed_fx_dof: (an integer (int or long))
        dof for fixed effects analysis
        argument: ``--ffxdof %d``
        mutually_exclusive: fixed_fx_dof_file
diag: (an integer (int or long))
        Gdiag_no : set diagnositic level
        argument: ``--diag %d``
sim_sign: ('abs' or 'pos' or 'neg')
        abs, pos, or neg
        argument: ``--sim-sign %s``
fixed_fx_var: (an existing file name)
        for fixed effects analysis
        argument: ``--yffxvar %s``
synth: (a boolean)
        replace input with gaussian
        argument: ``--synth``
weighted_ls: (an existing file name)
        weighted least squares
        argument: ``--wls %s``
        mutually_exclusive: weight_file, weight_inv, weight_sqrt
per_voxel_reg: (a list of items which are an existing file name)
        per-voxel regressors
        argument: ``--pvr %s...``
one_sample: (a boolean)
        construct X and C as a one-sample group mean
        argument: ``--osgm``
        mutually_exclusive: one_sample, fsgd, design, contrast
pca: (a boolean)
        perform pca/svd analysis on residual
        argument: ``--pca``
fsgd: (a tuple of the form: (an existing file name, 'doss' or
        'dods'))
```

(continues on next page)

(continued from previous page)

```

    freesurfer descriptor file
    argument: ``--fsgd %s %s``
    mutually_exclusive: fsgd, design, one_sample
diag_cluster: (a boolean)
    save sig volume and exit from first sim loop
    argument: ``--diag-cluster``
weight_file: (an existing file name)
    weight for each input at each voxel
    mutually_exclusive: weighted_ls
sim_done_file: (a file name)
    create file when simulation finished
    argument: ``--sim-done %s``
save_cond: (a boolean)
    flag to save design matrix condition at each voxel
    argument: ``--save-cond``
subject_id: (a unicode string)
    subject id for surface geometry
save_res_corr_mtx: (a boolean)
    save residual error spatial correlation matrix (eres.scm). Big!
    argument: ``--eres-scm``
uniform: (a tuple of the form: (a float, a float))
    use uniform distribution instead of gaussian
    argument: ``--uniform %f %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
calc_AR1: (a boolean)
    compute and save temporal AR1 of residual
    argument: ``--tar1``
compute_log_y: (a boolean)
    compute natural log of y prior to analysis
    argument: ``--logy``
save_residual: (a boolean)
    save residual error (eres)
    argument: ``--eres-save``
debug: (a boolean)
    turn on debugging
    argument: ``--debug``
allow_repeated_subjects: (a boolean)
    allow subject names to repeat in the fsgd file (must appear before
    --fsgd
    argument: ``--allowsubjrep``
save_estimate: (a boolean)
    save signal estimate (yhat)
    argument: ``--yhat-save``
resynth_test: (an integer (int or long))
    test GLM by resynthisis
    argument: ``--resynthtest %d``
design: (an existing file name)
    design matrix file
    argument: ``--X %s``
    mutually_exclusive: fsgd, design, one_sample
check_opts: (a boolean)
    don't run anything, just check options and exit
    argument: ``--checkopts``
surf: (a boolean)
    analysis is on a surface mesh

```

(continues on next page)

(continued from previous page)

```

        argument: ``--surf %s %s %s``
        requires: subject_id, hemi
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
prune: (a boolean)
        remove voxels that do not have a non-zero value at each frame (def)
        argument: ``--prune``
weight_inv: (a boolean)
        invert weights
        argument: ``--w-inv``
        mutually_exclusive: weighted_ls
no_mask_smooth: (a boolean)
        do not mask when smoothing
        argument: ``--no-mask-smooth``
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
        argument: ``--voxdump %d %d %d``
surf_geo: (a unicode string, nipy default value: white)
        surface geometry name (e.g. white, pial)
prune_thresh: (a float)
        prune threshold. Default is FLT_MIN
        argument: ``--prune_thr %f``
        mutually_exclusive: noprun
no_prune: (a boolean)
        do not prune
        argument: ``--no-prune``
        mutually_exclusive: prunethresh
weight_sqrt: (a boolean)
        sqrt of weights
        argument: ``--w-sqrt``
        mutually_exclusive: weighted_ls
profile: (an integer (int or long))
        niters : test speed
        argument: ``--profile %d``
no_contrast_ok: (a boolean)
        do not fail if no contrasts specified
        argument: ``--no-contrasts-ok``
allow_ill_cond: (a boolean)
        allow ill-conditioned design matrices
        argument: ``--illcond``
subjects_dir: (an existing directory name)
        subjects directory
label_file: (an existing file name)
        use label as mask, surfaces only
        argument: ``--label %s``
        mutually_exclusive: cortex
glm_dir: (a unicode string)
        save outputs to dir
        argument: ``--glmdir %s``
seed: (an integer (int or long))
        used for synthesizing noise
        argument: ``--seed %d``
fixed_fx_dof_file: (a file name)
        text file with dof for fixed effects analysis

```

(continues on next page)

(continued from previous page)

```

        argument: ``--ffxdofdat %d``
        mutually_exclusive: fixed_fx_dof
hemi: ('lh' or 'rh')
        surface hemisphere
mask_file: (an existing file name)
        binary mask
        argument: ``--mask %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
        argument: ``--sim %s %d %f %s``
no_est_fwhm: (a boolean)
        turn off FWHM output estimation
        argument: ``--no-est-fwhm``
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
        argument: ``--selfreg %d %d %d``
fwhm: (a floating point number >= 0.0)
        smooth input by fwhm
        argument: ``--fwhm %f``
force_perm: (a boolean)
        force permutation test, even when design matrix is not orthog
        argument: ``--perm-force``
var_fwhm: (a floating point number >= 0.0)
        smooth variance by fwhm
        argument: ``--var-fwhm %f``

```

Outputs:

```

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
svd_stats_file: (a file name)
        text file summarizing the residual PCA
gamma_file: (a list of items which are any value)
        map of contrast of regression coefficients
glm_dir: (an existing directory name)
        output directory
singular_values: (a file name)
        matrix singular values from residual PCA
beta_file: (an existing file name)
        map of regression coefficients
fwhm_file: (a file name)
        text file with estimated smoothness
sig_file: (a list of items which are any value)
        map of F-test significance (in -log10p)
mask_file: (a file name)
        map of the mask used in the analysis
dof_file: (a file name)
        text file with effective degrees-of-freedom for the analysis
estimate_file: (a file name)
        map of the estimated Y values
ftest_file: (a list of items which are any value)

```

(continues on next page)

(continued from previous page)

```

    map of test statistic values
gamma_var_file: (a list of items which are any value)
    map of regression contrast variance
spatial_eigenvectors: (a file name)
    map of spatial eigenvectors from residual PCA
frame_eigenvectors: (a file name)
    matrix of frame eigenvectors from residual PCA

```

62.2.11 SegStats

[Link to code](#)

Wraps the executable 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]
surf_label: (a tuple of the form: (a unicode string, 'lh' or 'rh', a
    unicode string))
    subject hemi label : use surface label
    argument: ``--slabel %s %s %s``
    mutually_exclusive: segmentation_file, annot, surf_label
annot: (a tuple of the form: (a unicode string, 'lh' or 'rh', a
    unicode string))
    subject hemi parc : use surface parcellation
    argument: ``--annot %s %s %s``
    mutually_exclusive: segmentation_file, annot, surf_label
segmentation_file: (an existing file name)
    segmentation volume path
    argument: ``--seg %s``
    mutually_exclusive: segmentation_file, annot, surf_label

[Optional]
avgwf_file: (a boolean or a file name)
    Save as binary volume (bool or filename)
    argument: ``--avgwfvol %s``
default_color_table: (a boolean)
    use $FREESURFER_HOME/FreeSurferColorLUT.txt
    argument: ``--ctab-default``
    mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
exclude_id: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

        Exclude seg id from report
        argument: ``--excludeid %d``
partial_volume_file: (an existing file name)
        Compensate for partial voluming
        argument: ``--pv %s``
frame: (an integer (int or long))
        Report stats on nth frame of input volume
        argument: ``--frame %d``
calc_power: ('sqr' or 'sqrt')
        Compute either the sqr or the sqrt of the input
        argument: ``--%s``
total_gray: (a boolean)
        Compute volume of total gray matter
        argument: ``--totalgray``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
mask_erode: (an integer (int or long))
        Erode mask by some amount
        argument: ``--maskerode %d``
color_table_file: (an existing file name)
        color table file with seg id names
        argument: ``--ctab %s``
        mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
brain_vol: ('brain-vol-from-seg' or 'brainmask')
        Compute brain volume either with ``brainmask`` or ``brain-vol-from-
        seg``
        argument: ``--%s``
calc_snr: (a boolean)
        save mean/std as extra column in output table
        argument: ``--snr``
supratent: (a boolean)
        Undocumented input flag
        argument: ``--supratent``
empty: (a boolean)
        Report on segmentations listed in the color table
        argument: ``--empty``
wm_vol_from_surf: (a boolean)
        Compute wm volume from surf
        argument: ``--surf-wm-vol``
etiv_only: ('etiv' or 'old-etiv' or '--s-only')
        Compute etiv and exit. Use ``etiv`` or ``old-etiv``
exclude_ctx_gm_wm: (a boolean)
        exclude cortical gray and white matter
        argument: ``--excl-ctxgmwm``
in_intensity: (a file name)
        Undocumented input norm.mgz file
        argument: ``--in %s --in-intensity-name %s``
subcort_gm: (a boolean)
        Compute volume of subcortical gray matter
        argument: ``--subcortgray``
subjects_dir: (an existing directory name)
        subjects directory
args: (a unicode string)
        Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

    argument: ``%s``
segment_id: (a list of items which are any value)
    Manually specify segmentation ids
    argument: ``--id %s...``
mask_frame: (an integer (int or long))
    Mask with this (0 based) frame of the mask volume
    requires: mask_file
multiply: (a float)
    multiply input by val
    argument: ``--mul %f``
summary_file: (a file name)
    Segmentation stats summary table file
    argument: ``--sum %s``, position: -1
gca_color_table: (an existing file name)
    get color table from GCA (CMA)
    argument: ``--ctab-gca %s``
    mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
non_empty_only: (a boolean)
    Only report nonempty segmentations
    argument: ``--nonempty``
etiv: (a boolean)
    Compute ICV from talairach transform
    argument: ``--etiv``
avgwf_txt_file: (a boolean or a file name)
    Save average waveform into file (bool or filename)
    argument: ``--avgwf %s``
sf_avg_file: (a boolean or a file name)
    Save mean across space and time
    argument: ``--sfavg %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
    argument: ``--brainmask %s``
mask_thresh: (a float)
    binarize mask with this threshold <0.5>
    argument: ``--maskthresh %f``
mask_sign: ('abs' or 'pos' or 'neg' or '--masksign %s')
    Sign for mask threshold: pos, neg, or abs
mask_file: (an existing file name)
    Mask volume (same size as seg
    argument: ``--mask %s``
intensity_units: ('MR')
    Intensity units
    argument: ``--in-intensity-units %s``
    requires: in_intensity
mask_invert: (a boolean)
    Invert binarized mask volume
    argument: ``--maskinvert``
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)
    argument: ``--vox %s``
euler: (a boolean)
    Write out number of defect holes in orig.nofix based on the euler
    number
    argument: ``--euler``
in_file: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

        Use the segmentation to report stats on this volume
        argument: ``--i %s``
cortex_vol_from_surf: (a boolean)
        Compute cortex volume from surf
        argument: ``--surf-ctx-vol``

```

Outputs:

```

avgwf_file: (a file name)
        Volume 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
avgwf_txt_file: (a file name)
        Text file with functional statistics averaged over segs

```

62.2.12 SegStatsReconAll[Link to code](#)Wraps the executable 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]
rh_pial: (an existing file name)
    Input file must be <subject_id>/surf/rh.pial
ribbon: (a file name)
    Input file mri/ribbon.mgz
transform: (an existing file name)
    Input transform file
rh_orig_nofix: (an existing file name)
    Input rh.orig.nofix
segmentation_file: (an existing file name)
    segmentation volume path
    argument: ``--seg %s``
    mutually_exclusive: segmentation_file, annot, surf_label
rh_white: (an existing file name)
    Input file must be <subject_id>/surf/rh.white
lh_orig_nofix: (an existing file name)
    Input lh.orig.nofix
subject_id: (a string, nipy default value: subject_id)
    Subject id being processed
    argument: ``--subject %s``
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
annot: (a tuple of the form: (a unicode string, 'lh' or 'rh', a
    unicode string))
    subject hemi parc : use surface parcellation
    argument: ``--annot %s %s %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
    argument: ``--slabel %s %s %s``
    mutually_exclusive: segmentation_file, annot, surf_label

[Optional]
presurf_seg: (an existing file name)
    Input segmentation volume
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)
    argument: ``--avgwfvol %s``
exclude_id: (an integer (int or long))
    Exclude seg id from report
    argument: ``--excludeid %d``
sf_avg_file: (a boolean or a file name)
    Save mean across space and time
    argument: ``--sfavg %s``
color_table_file: (an existing file name)
    color table file with seg id names
    argument: ``--ctab %s``
    mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
mask_thresh: (a float)
    binarize mask with this threshold <0.5>

```

(continues on next page)

(continued from previous page)

```

        argument: ``--maskthresh %f``
empty: (a boolean)
    Report on segmentations listed in the color table
    argument: ``--empty``
wm_vol_from_surf: (a boolean)
    Compute wm volume from surf
    argument: ``--surf-wm-vol``
subcort_gm: (a boolean)
    Compute volume of subcortical gray matter
    argument: ``--subcortgray``
segment_id: (a list of items which are any value)
    Manually specify segmentation ids
    argument: ``--id %s...``
mask_frame: (an integer (int or long))
    Mask with this (0 based) frame of the mask volume
    requires: mask_file
multiply: (a float)
    multiply input by val
    argument: ``--mul %f``
gca_color_table: (an existing file name)
    get color table from GCA (CMA)
    argument: ``--ctab-gca %s``
    mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
non_empty_only: (a boolean)
    Only report nonempty segmentations
    argument: ``--nonempty``
etiv: (a boolean)
    Compute ICV from talairach transform
    argument: ``--etiv``
copy_inputs: (a boolean)
    If running as a node, set this to True otherwise, this will copy the
    implicit inputs to the node directory.
calc_power: ('sqr' or 'sqrt')
    Compute either the sqr or the sqrt of the input
    argument: ``--%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
    argument: ``--brainmask %s``
mask_invert: (a boolean)
    Invert binarized mask volume
    argument: ``--maskinvert``
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)
    argument: ``--vox %s``
intensity_units: ('MR')
    Intensity units
    argument: ``--in-intensity-units %s``
    requires: in_intensity
default_color_table: (a boolean)
    use $FREESURFER_HOME/FreeSurferColorLUT.txt
    argument: ``--ctab-default``
    mutually_exclusive: color_table_file, default_color_table,
        gca_color_table
exclude_ctx_gm_wm: (a boolean)
    exclude cortical gray and white matter

```

(continues on next page)

(continued from previous page)

```

        argument: ``--excl-ctxgwmw``
summary_file: (a file name)
    Segmentation stats summary table file
    argument: ``--sum %s``, position: -1
partial_volume_file: (an existing file name)
    Compensate for partial voluming
    argument: ``--pv %s``
frame: (an integer (int or long))
    Report stats on nth frame of input volume
    argument: ``--frame %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
mask_erode: (an integer (int or long))
    Erode mask by some amount
    argument: ``--maskerode %d``
brain_vol: ('brain-vol-from-seg' or 'brainmask')
    Compute brain volume either with ``brainmask`` or ``brain-vol-from-
    seg``
    argument: ``--%s``
calc_snr: (a boolean)
    save mean/std as extra column in output table
    argument: ``--snr``
etiv_only: ('etiv' or 'old-etiv' or '--s-only')
    Compute etiv and exit. Use ``etiv`` or ``old-etiv``
in_intensity: (a file name)
    Undocumented input norm.mgz file
    argument: ``--in %s --in-intensity-name %s``
in_file: (an existing file name)
    Use the segmentation to report stats on this volume
    argument: ``--i %s``
subjects_dir: (an existing directory name)
    subjects directory
supratent: (a boolean)
    Undocumented input flag
    argument: ``--supratent``
mask_sign: ('abs' or 'pos' or 'neg' or '--masksign %s')
    Sign for mask threshold: pos, neg, or abs
mask_file: (an existing file name)
    Mask volume (same size as seg
    argument: ``--mask %s``
euler: (a boolean)
    Write out number of defect holes in orig.nofix based on the euler
    number
    argument: ``--euler``
cortex_vol_from_surf: (a boolean)
    Compute cortex volume from surf
    argument: ``--surf-ctx-vol``
total_gray: (a boolean)
    Compute volume of total gray matter
    argument: ``--totalgray``
avgwf_txt_file: (a boolean or a file name)
    Save average waveform into file (bool or filename)
    argument: ``--avgwf %s``
args: (a unicode string)
    Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```
argument: ``%s``
```

Outputs:

```
avgwf_file: (a file name)
    Volume 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
avgwf_txt_file: (a file name)
    Text file with functional statistics averaged over segs
```

62.2.13 SphericalAverage[Link to code](#)Wraps the executable command `mris_spherical_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]
subject_id: (a string)
    Output subject id
    argument: ``-o %s``
in_surf: (an existing file name)
    Input surface file
    argument: ``%s``, position: -3
hemisphere: ('lh' or 'rh')
    Input hemisphere
    argument: ``%s``, position: -4
which: ('coords' or 'label' or 'vals' or 'curv' or 'area')
    No documentation
    argument: ``%s``, position: -6
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
```

(continues on next page)

(continued from previous page)

```

        '{in_average}/label/rh.entorhinal.label'
        argument: ``%s``, position: -5

[Optional]
out_file: (a file name)
        Output filename
        argument: ``%s``, position: -1
in_average: (a directory name)
        Average subject
        argument: ``%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
in_orig: (an existing file name)
        Original surface filename
        argument: ``-orig %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
threshold: (a float)
        Undocumented
        argument: ``-t %.1f``
erode: (an integer (int or long))
        Undocumented
        argument: ``-erode %d``
subjects_dir: (an existing directory name)
        subjects directory

```

Outputs:

```

out_file: (a file name)
        Output label

```

62.3 interfaces.freesurfer.preprocess

62.3.1 ApplyVolTransform

[Link to code](#)Wraps the executable 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]
tal: (a boolean)
    map to a sub FOV of MNI305 (with --reg only)
    argument: ``--tal``
    mutually_exclusive: target_file, tal, fs_target
xfm_reg_file: (an existing file name)
    ScannerRAS-to-ScannerRAS matrix (MNI format)
    argument: ``--xfm %s``
    mutually_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,
        xfm_reg_file, reg_header, mni_152_reg, subject
lta_file: (an existing file name)
    Linear Transform Array file
    argument: ``--lta %s``
    mutually_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,
        xfm_reg_file, reg_header, mni_152_reg, subject
lta_inv_file: (an existing file name)
    LTA, invert
    argument: ``--lta-inv %s``
    mutually_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,
        xfm_reg_file, reg_header, mni_152_reg, subject
reg_header: (a boolean)
    ScannerRAS-to-ScannerRAS matrix = identity
    argument: ``--regheader``
    mutually_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,
        xfm_reg_file, reg_header, mni_152_reg, subject
source_file: (an existing file name)
    Input volume you wish to transform
    argument: ``--mov %s``
target_file: (an existing file name)
    Output template volume
    argument: ``--targ %s``
    mutually_exclusive: target_file, tal, fs_target
mni_152_reg: (a boolean)
    target MNI152 space
    argument: ``--regheader``
    mutually_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,
        xfm_reg_file, reg_header, mni_152_reg, subject
fsl_reg_file: (an existing file name)
    fslRAS-to-fslRAS matrix (FSL format)
    argument: ``--fsl %s``
    mutually_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,
        xfm_reg_file, reg_header, mni_152_reg, subject
fs_target: (a boolean)
    use orig.mgz from subject in regfile as target
    argument: ``--fstarg``
    mutually_exclusive: target_file, tal, fs_target
    requires: reg_file
reg_file: (an existing file name)
    tkRAS-to-tkRAS matrix (tkregister2 format)
    argument: ``--reg %s``
    mutually_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,
        xfm_reg_file, reg_header, mni_152_reg, subject
subject: (a unicode string)
    set matrix = identity and use subject for any templates
    argument: ``--s %s``
    mutually_exclusive: reg_file, lta_file, lta_inv_file, fsl_reg_file,
        xfm_reg_file, reg_header, mni_152_reg, subject

```

(continues on next page)

(continued from previous page)

```

[Optional]
no_resample: (a boolean)
    Do not resample; just change vox2ras matrix
    argument: ``--no-resample``
transformed_file: (a file name)
    Output volume
    argument: ``--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
interp: ('trilin' or 'nearest' or 'cubic')
    Interpolation method (<trilin> or nearest)
    argument: ``--interp %s``
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.
    argument: ``--m3z %s``
inverse: (a boolean)
    sample from target to source
    argument: ``--inv``
subjects_dir: (an existing directory name)
    subjects directory
tal_resolution: (a float)
    Resolution to sample when using tal
    argument: ``--talres %.10f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
invert_morph: (a boolean)
    Compute and use the inverse of the non-linear morph to resample the
    input volume. To be used by --m3z.
    argument: ``--inv-morph``
    requires: m3z_file
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.
    argument: ``--noDefM3zPath``
    requires: m3z_file

```

Outputs:

```

transformed_file: (an existing file name)
    Path to output file if used normally

```

62.3.2 BBRegister[Link to code](#)Wraps the executable command `bbregister`.Use FreeSurfer `bbregister` 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]
subject_id: (a unicode string)
    freesurfer subject id
    argument: ``--s %s``
contrast_type: ('t1' or 't2' or 'bold' or 'dti')
    contrast type of image
    argument: ``--%s``
source_file: (a file name)
    source file to be registered
    argument: ``--mov %s``

[Optional]
init: ('coreg' or 'rr' or 'spm' or 'fsl' or 'header' or 'best')
    initialize registration with mri_coreg, spm, fsl, or header
    argument: ``--init-%s``
    mutually_exclusive: init_reg_file
init_cost_file: (a boolean or a file name)
    output initial registration cost file
    argument: ``--initcost %s``
spm_nifti: (a boolean)
    force use of nifti rather than analyze with SPM
    argument: ``--spm-nii``
subjects_dir: (an existing directory name)
    subjects directory
epi_mask: (a boolean)
    mask out B0 regions in stages 1 and 2
    argument: ``--epi-mask``
reg_frame: (an integer (int or long))
    0-based frame index for 4D source file
    argument: ``--frame %d``
    mutually_exclusive: reg_middle_frame
reg_middle_frame: (a boolean)
    Register middle frame of 4D source file
    argument: ``--mid-frame``
    mutually_exclusive: reg_frame
out_reg_file: (a file name)
    output registration file
    argument: ``--reg %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
registered_file: (a boolean or a file name)
    output warped sourcefile either True or filename
    argument: ``--o %s``
init_reg_file: (an existing file name)
    existing registration file
```

(continues on next page)

(continued from previous page)

```

        argument: ``--init-reg %s``
        mutually_exclusive: init
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
intermediate_file: (an existing file name)
    Intermediate image, e.g. in case of partial FOV
    argument: ``--int %s``
out_lta_file: (a boolean or a file name)
    write the transformation matrix in LTA format
    argument: ``--lta %s``
dof: (6 or 9 or 12)
    number of transform degrees of freedom
    argument: ``--%d``
out_fsl_file: (a boolean or a file name)
    write the transformation matrix in FSL FLIRT format
    argument: ``--fslmat %s``
fsl_dof: (an integer (int or long))
    degrees of freedom for initial registration (FSL)
    argument: ``--fsl-dof %d``

```

Outputs:

```

init_cost_file: (an existing file name)
    Output initial registration cost file
registered_file: (an existing file name)
    Registered and resampled source file
min_cost_file: (an existing file name)
    Output registration minimum cost file
out_reg_file: (an existing file name)
    Output registration file
out_fsl_file: (an existing file name)
    Output FLIRT-style registration file
out_lta_file: (an existing file name)
    Output LTA-style registration file

```

62.3.3 CALabel[Link to code](#)Wraps the executable 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]
transform: (an existing file name)
    Input transform for CALabel
    argument: ``%s``, position: -3
in_file: (an existing file name)
    Input volume for CALabel
    argument: ``%s``, position: -4
template: (an existing file name)
    Input template for CALabel
    argument: ``%s``, position: -2
out_file: (a file name)
    Output file for CALabel
    argument: ``%s``, position: -1

[Optional]
aseg: (a file name)
    Undocumented flag. Autorecon3 uses ../mri/aseg.presurf.mgz as input
    file
    argument: ``-aseg %s``
in_vol: (an existing file name)
    set input volume
    argument: ``-r %s``
label: (a file name)
    Undocumented flag. Autorecon3 uses
    ../label/{hemisphere}.cortex.label as input file
    argument: ``-l %s``
subjects_dir: (an existing directory name)
    subjects directory
num_threads: (an integer (int or long))
    allows for specifying more threads
align: (a boolean)
    Align CALabel
    argument: ``-align``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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
    argument: ``-relabel_unlikely %d %.1f``
intensities: (an existing file name)
    input label intensities file(used in longitudinal processing)
    argument: ``-r %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
no_big_ventricles: (a boolean)
    No big ventricles
    argument: ``-nobigventricles``
prior: (a float)
    Prior for CALabel
    argument: ``-prior %.1f``

```

Outputs:

```
out_file: (a file name)
          Output volume from CALabel
```

62.3.4 CANormalize

[Link to code](#)

Wraps the executable 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]
transform: (an existing file name)
            The tranform file in lta format
            argument: ``%s``, position: -2
atlas: (an existing file name)
        The atlas file in gca format
        argument: ``%s``, position: -3
in_file: (an existing file name)
        The input file for CANormalize
        argument: ``%s``, position: -4

[Optional]
mask: (an existing file name)
      Specifies volume to use as mask
      argument: ``-mask %s``
subjects_dir: (an existing directory name)
              subjects directory
out_file: (a file name)
          The output file for CANormalize
          argument: ``%s``, position: -1
control_points: (a file name)
                File name for the output control points
                argument: ``-c %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
long_file: (a file name)
           undocumented flag used in longitudinal processing
           argument: ``-long %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```

out_file: (a file name)
    The output file for Normalize
control_points: (a file name)
    The output control points for Normalize

```

62.3.5 CRegister

[Link to code](#)

Wraps the executable command `mri_ca_register`.

Generates a multi-dimensional talairach transform from a gca file and talairach.lta file

For complete details, see the [FS Documentation](#)

Examples

```

>>> from nipy.interfaces import freesurfer
>>> ca_register = freesurfer.CRegister()
>>> 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 CRegister
    argument: ``%s``, position: -3

[Optional]
mask: (an existing file name)
    Specifies volume to use as mask
    argument: ``-mask %s``
subjects_dir: (an existing directory name)
    subjects directory
A: (an integer (int or long))
    undocumented flag used in longitudinal processing
    argument: ``-A %d``
transform: (an existing file name)
    Specifies transform in lta format
    argument: ``-T %s``
num_threads: (an integer (int or long))
    allows for specifying more threads
align: (a string)
    Specifies when to perform alignment
    argument: ``-align-%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
l_files: (a list of items which are a file name)
    undocumented flag used in longitudinal processing
    argument: ``-l %s``
template: (an existing file name)
    The template file in gca format
    argument: ``%s``, position: -2
levels: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

        defines how many surrounding voxels will be used in interpolations,
        default is 6
        argument: ``-levels %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
no_big_ventricles: (a boolean)
    No big ventricles
    argument: ``-nobigventricles``
invert_and_save: (a boolean)
    Invert and save the .m3z multi-dimensional talarach transform to x,
    y, and z .mgz files
    argument: ``-invert-and-save``, position: -4
out_file: (a file name)
    The output volume for CRegister
    argument: ``%s``, position: -1

```

Outputs:

```

out_file: (a file name)
    The output file for CRegister

```

62.3.6 ConcatenateLTA[Link to code](#)Wraps the executable 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 = 'lta1.lta'
>>> conc_lta.inputs.in_lta2 = 'lta2.lta'
>>> conc_lta.cmdline
'mri_concatenate_lta lta1.lta lta2.lta lta1_concat.lta'

```

You can use 'identity.nofile' as the filename for `in_lta2`, e.g.:

```

>>> conc_lta.inputs.in_lta2 = 'identity.nofile'
>>> conc_lta.inputs.invert_1 = True
>>> conc_lta.inputs.out_file = 'inv1.lta'
>>> conc_lta.cmdline
'mri_concatenate_lta -invert1 lta1.lta identity.nofile inv1.lta'

```

To create a RAS2RAS transform:

```

>>> conc_lta.inputs.out_type = 'RAS2RAS'
>>> conc_lta.cmdline
'mri_concatenate_lta -invert1 -out_type 1 lta1.lta identity.nofile inv1.lta'

```

Inputs:

```

[Mandatory]
in_lta2: (an existing file name or 'identity.nofile')
    maps dst1(src2) to dst2

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``, position: -2
in_lta1: (an existing file name)
        maps some src1 to dst1
        argument: ``%s``, position: -3

[Optional]
invert_1: (a boolean)
        invert in_lta1 before applying it
        argument: ``-invert1``
out_file: (a file name)
        the combined LTA maps: src1 to dst2 = LTA2*LTA1
        argument: ``%s``, position: -1
out_type: ('VOX2VOX' or 'RAS2RAS')
        set final LTA type
        argument: ``-out_type %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
invert_out: (a boolean)
        invert output LTA
        argument: ``-invertout``
tal_source_file: (a file name)
        if in_lta2 is talairach.xfm, specify source for talairach
        argument: ``-tal %s``, position: -5
        requires: tal_template_file
invert_2: (a boolean)
        invert in_lta2 before applying it
        argument: ``-invert2``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
tal_template_file: (a file name)
        if in_lta2 is talairach.xfm, specify template for talairach
        argument: ``%s``, position: -4
        requires: tal_source_file
subject: (a unicode string)
        set subject in output LTA
        argument: ``-subject %s``
subjects_dir: (an existing directory name)
        subjects directory

```

Outputs:

```

out_file: (a file name)
        the combined LTA maps: src1 to dst2 = LTA2*LTA1

```

62.3.7 DICOMConvert[Link to code](#)

Wraps the executable 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]
dicom_dir: (an existing directory name)
    dicom directory from which to convert dicom files
base_output_dir: (a directory name)
    directory in which subject directories are created

[Optional]
dicom_info: (an existing file name)
    File containing summary information from mri_parse_sdcmdir
seq_list: (a list of items which are a unicode string)
    list of pulse sequence names to be converted.
    requires: dicom_info
subjects_dir: (an existing directory name)
    subjects directory
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
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
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
subject_id: (any value)
    subject identifier to insert into template
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
ignore_single_slice: (a boolean)
    ignore volumes containing a single slice
    requires: dicom_info
subject_dir_template: (a unicode string, nipy default value:
    S.%04d)
    template for subject directory name
```

Outputs:

None

62.3.8 EditWMwithAseg

[Link to code](#)

Wraps the executable 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]
seg_file: (an existing file name)
    Input presurf segmentation file
    argument: ``%s``, position: -2
in_file: (an existing file name)
    Input white matter segmentation file
    argument: ``%s``, position: -4
brain_file: (an existing file name)
    Input brain/T1 file
    argument: ``%s``, position: -3
out_file: (a file name)
    File to be written as output
    argument: ``%s``, position: -1

[Optional]
keep_in: (a boolean)
    Keep edits as found in input volume
    argument: ``-keep-in``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
subjects_dir: (an existing directory name)
    subjects directory
```

Outputs:

```
out_file: (a file name)
    Output edited WM file
```

62.3.9 FitMSParams

[Link to code](#)

Wraps the executable command `mri_ms_fitparms`.
Estimate tissue parameters from a set of FLASH images.

Examples

```
>>> from nipy.interfaces.freesurfer import FitMSParams
>>> msfit = FitMSParams()
```

(continues on next page)

(continued from previous page)

```
>>> 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)
          argument: ``%s``, position: -2

[Optional]
tr_list: (a list of items which are an integer (int or long))
          list of TRs of the input files (in msec)
te_list: (a list of items which are a float)
          list of TEs of the input files (in msec)
subjects_dir: (an existing directory name)
              subjects directory
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
out_dir: (a directory name)
          directory to store output in
          argument: ``%s``, position: -1
xfm_list: (a list of items which are an existing file name)
          list of transform files to apply to each FLASH image
flip_list: (a list of items which are an integer (int or long))
           list of flip angles of the input 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', nipype default value: {})
          Environment variables
```

Outputs:

```
t2star_image: (an existing file name)
              image of estimated T2* values
pd_image: (an existing file name)
           image of estimated proton density values
t1_image: (an existing file name)
           image of estimated T1 relaxation values
```

62.3.10 MNIBiasCorrection[Link to code](#)

Wraps the executable 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.
        argument: ``--i %s``

[Optional]
no_rescale: (a boolean)
            do not rescale so that global mean of output == input global mean
            argument: ``--no-rescale``
mask: (an existing file name)
      brainmask volume. Input can be any format accepted by mri_convert.
      argument: ``--mask %s``
distance: (an integer (int or long))
          N3 -distance option
          argument: ``--distance %d``
subjects_dir: (an existing directory name)
              subjects directory
transform: (an existing file name)
           tal.xfm. Use mri_make_uchar instead of conforming
           argument: ``--uchar %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
stop: (a float)
      Convergence threshold below which iteration stops (suggest 0.01 to
      0.0001)
      argument: ``--stop %f``
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.
                    argument: ``--proto-iters %d``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
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.
```

(continues on next page)

(continued from previous page)

```

        argument: ``--o %s``
iterations: (an integer (int or long), nipy default value: 4)
    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.
        argument: ``--n %d``
shrink: (an integer (int or long))
    Shrink parameter for finer sampling (default is 4)
        argument: ``--shrink %d``

```

Outputs:

```

out_file: (an existing file name)
    output volume

```

62.3.11 MRIConvert[Link to code](#)

Wraps the executable 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
    argument: ``--input_volume %s``, position: -2

[Optional]
out_k_count: (an integer (int or long))
    some count ?? in k direction
    argument: ``--out_k_count %d``
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 'nifti1' or 'nii' or
    'niigz' or 'ge' or 'gelx' or 'lx' or 'ximg' or 'siemens' or
    'dicom' or 'siemens_dicom')
    template file type
    argument: ``--template_type %s``
out_i_count: (an integer (int or long))
    some count ?? in i direction
    argument: ``--out_i_count %d``

```

(continues on next page)

(continued from previous page)

```

unwarp_gradient: (a boolean)
    unwarp gradient nonlinearity
    argument: ``--unwarp_gradient_nonlinearity``
read_only: (a boolean)
    read the input volume
    argument: ``--read_only``
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)
    argument: ``--fsubsample %d %d %d``
conform: (a boolean)
    conform to 1mm voxel size in coronal slice direction with 256^3 or
    more
    argument: ``--conform``
out_j_size: (an integer (int or long))
    output j size
    argument: ``--out_j_size %d``
in_scale: (a float)
    input intensity scale factor
    argument: ``--scale %f``
crop_gdf: (a boolean)
    apply GDF cropping
    argument: ``--crop_gdf``
in_j_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    argument: ``--in_j_direction %f %f %f``
cut_ends: (an integer (int or long))
    remove ncut slices from the ends
    argument: ``--cutends %d``
out_file: (a file name)
    output filename or True to generate one
    argument: ``--output_volume %s``, position: -1
out_stats: (a boolean)
    display output stats
    argument: ``--out_stats``
status_file: (a file name)
    status file for DICOM conversion
    argument: ``--status %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
cw256: (a boolean)
    conform to dimensions of 256^3
    argument: ``--cw256``
in_stats: (a boolean)
    display input stats
    argument: ``--in_stats``
force_ras: (a boolean)
    use default when orientation info absent
    argument: ``--force_ras_good``
in_k_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    argument: ``--in_k_direction %f %f %f``
no_translate: (a boolean)
    ~~~
    argument: ``--no_translate``
out_datatype: ('uchar' or 'short' or 'int' or 'float')

```

(continues on next page)

(continued from previous page)

```

        output data type <uchar|short|int|float>
        argument: ``--out_data_type %s``
color_file: (an existing file name)
        color file
        argument: ``--color_file %s``
slice_reverse: (a boolean)
        reverse order of slices, update vox2ras
        argument: ``--slice-reverse``
fill_parcellation: (a boolean)
        fill parcellation
        argument: ``--fill_parcellation``
split: (a boolean)
        split output frames into separate output files.
        argument: ``--split``
out_i_size: (an integer (int or long))
        output i size
        argument: ``--out_i_size %d``
in_center: (a list of at most 3 items which are a float)
        <R coordinate> <A coordinate> <S coordinate>
        argument: ``--in_center %s``
autoalign_matrix: (an existing file name)
        text file with autoalign matrix
        argument: ``--autoalign %s``
zero_outlines: (a boolean)
        zero outlines
        argument: ``--zero_outlines``
out_center: (a tuple of the form: (a float, a float, a float))
        <R coordinate> <A coordinate> <S coordinate>
        argument: ``--out_center %f %f %f``
sdcm_list: (an existing file name)
        list of DICOM files for conversion
        argument: ``--sdcm_list %s``
in_i_dir: (a tuple of the form: (a float, a float, a float))
        <R direction> <A direction> <S direction>
        argument: ``--in_i_direction %f %f %f``
no_change: (a boolean)
        don't change type of input to that of template
        argument: ``--nochange``
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
        argument: ``--out_orientation %s``
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>
        argument: ``--cropsize %d %d %d``
invert_contrast: (a float)
        threshold for inversting contrast
        argument: ``--invert_contrast %f``
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

```

(continues on next page)

(continued from previous page)

```

        argument: ``-voysize %f %f %f``
conform_min: (a boolean)
    conform to smallest size
    argument: ``--conform_min``
sphinx: (a boolean)
    change orientation info to sphinx
    argument: ``--sphinx``
no_write: (a boolean)
    do not write output
    argument: ``--no_write``
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
    argument: ``--in_orientation %s``
skip_n: (an integer (int or long))
    skip the first n frames
    argument: ``--nskip %d``
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)
    argument: ``--crop %d %d %d``
out_k_size: (an integer (int or long))
    output k size
    argument: ``--out_k_size %d``
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
    argument: ``--out_type %s``
zero_ge_z_offset: (a boolean)
    zero ge z offset ???
    argument: ``--zero_ge_z_offset``
template_info: (a boolean)
    dump info about template
    argument: ``--template_info``
midframe: (a boolean)
    keep only the middle frame
    argument: ``--mid-frame``
apply_inv_transform: (an existing file name)
    apply inverse transformation xfm file
    argument: ``--apply_inverse_transform %s``
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
    argument: ``--slice-crop %d %d``
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

```

(continues on next page)

(continued from previous page)

```

        argument: ``--in_type %s``
slice_bias: (a float)
    apply half-cosine bias field
    argument: ``--slice-bias %f``
subject_name: (a unicode string)
    subject name ???
    argument: ``--subject_name %s``
in_info: (a boolean)
    display input info
    argument: ``--in_info``
te: (an integer (int or long))
    TE in msec
    argument: ``-te %d``
no_scale: (a boolean)
    dont rescale values for COR
    argument: ``--no_scale 1``
in_j_size: (an integer (int or long))
    input j size
    argument: ``--in_j_size %d``
out_scale: (a float)
    output intensity scale factor
    argument: ``--out-scale %d``
fwhm: (a float)
    smooth input volume by fwhm mm
    argument: ``--fwhm %f``
ascii: (a boolean)
    save output as ascii col>row>slice>frame
    argument: ``--ascii``
in_k_size: (an integer (int or long))
    input k size
    argument: ``--in_k_size %d``
resample_type: ('interpolate' or 'weighted' or 'nearest' or 'sinc' or
    'cubic')
    <interpolate|weighted|nearest|sinc|cubic> (default is interpolate)
    argument: ``--resample_type %s``
frame: (an integer (int or long))
    keep only 0-based frame number
    argument: ``--frame %d``
out_k_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    argument: ``--out_k_direction %f %f %f``
drop_n: (an integer (int or long))
    drop the last n frames
    argument: ``--ndrop %d``
devolve_transform: (a unicode string)
    subject id
    argument: ``--devolvexfm %s``
tr: (an integer (int or long))
    TR in msec
    argument: ``-tr %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
in_like: (an existing file name)
    input looks like
    argument: ``--in_like %s``

```

(continues on next page)

(continued from previous page)

```

out_i_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    argument: ``--out_i_direction %f %f %f``
in_i_size: (an integer (int or long))
    input i size
    argument: ``--in_i_size %d``
out_j_count: (an integer (int or long))
    some count ?? in j direction
    argument: ``--out_j_count %d``
parse_only: (a boolean)
    parse input only
    argument: ``--parse_only``
in_matrix: (a boolean)
    display input matrix
    argument: ``--in_matrix``
out_matrix: (a boolean)
    display output matrix
    argument: ``--out_matrix``
apply_transform: (an existing file name)
    apply xfm file
    argument: ``--apply_transform %s``
subjects_dir: (an existing directory name)
    subjects directory
out_info: (a boolean)
    display output info
    argument: ``--out_info``
reslice_like: (an existing file name)
    reslice output to match file
    argument: ``--reslice_like %s``
out_j_dir: (a tuple of the form: (a float, a float, a float))
    <R direction> <A direction> <S direction>
    argument: ``--out_j_direction %f %f %f``
smooth_parcellation: (a boolean)
    smooth parcellation
    argument: ``--smooth_parcellation``
ti: (an integer (int or long))
    TI in msec (note upper case flag)
    argument: ``-ti %d``
reorder: (a tuple of the form: (an integer (int or long), an integer
    (int or long), an integer (int or long)))
    olddim1 olddim2 olddim3
    argument: ``--reorder %d %d %d``
conform_size: (a float)
    conform to size_in_mm
    argument: ``--conform_size %s``

```

Outputs:

```

out_file: (a list of items which are an existing file name)
    converted output file

```

62.3.12 MRIsCAlabel[Link to code](#)Wraps the executable command `mris_ca_label`.

For a single subject, produces an annotation file, in which each cortical surface vertex is assigned a neuroanatomical 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.MRIsCLabel()
>>> 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]
classifier: (an existing file name)
    Classifier array input file
    argument: ``%s``, position: -2
subject_id: (a string, nipy default value: subject_id)
    Subject name or ID
    argument: ``%s``, position: -5
smoothwm: (an existing file name)
    implicit input {hemisphere}.smoothwm
canonsurf: (an existing file name)
    Input canonical surface file
    argument: ``%s``, position: -3
hemisphere: ('lh' or 'rh')
    Hemisphere ('lh' or 'rh')
    argument: ``%s``, position: -4
curv: (an existing file name)
    implicit input {hemisphere}.curv
sulc: (an existing file name)
    implicit input {hemisphere}.sulc

[Optional]
aseg: (a file name)
    Undocumented flag. Autorecon3 uses ../mri/aseg.presurf.mgz as input
    file
    argument: ``-aseg %s``
label: (a file name)
    Undocumented flag. Autorecon3 uses
    ../label/{hemisphere}.cortex.label as input file
    argument: ``-l %s``
num_threads: (an integer (int or long))
    allows for specifying more threads
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: {})
```

(continues on next page)

(continued from previous page)

```

    Environment variables
seed: (an integer (int or long))
    argument: ``-seed %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
out_file: (a file name)
    Annotated surface output file
    argument: ``%s``, position: -1
subjects_dir: (an existing directory name)
    subjects directory

```

Outputs:

```

out_file: (a file name)
    Output volume from MRIscALabel

```

62.3.13 Normalize[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -2

[Optional]
mask: (an existing file name)
    The input mask file for Normalize
    argument: ``-mask %s``
gradient: (an integer (int or long))
    use max intensity/mm gradient g (default=1)
    argument: ``-g %d``
subjects_dir: (an existing directory name)
    subjects directory
out_file: (a file name)
    The output file for Normalize
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
transform: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    Transform file from the header of the input 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', nipy default value: {})
    Environment variables
segmentation: (an existing file name)
    The input segmentation for Normalize
argument: ``-aseg %s``

```

Outputs:

```

out_file: (a file name)
    The output file for Normalize

```

62.3.14 ParseDICOMDir[Link to code](#)Wraps the executable 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
    argument: ``--d %s``

[Optional]
summarize: (a boolean)
    only print out info for run leaders
    argument: ``--summarize``
subjects_dir: (an existing directory name)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
dicom_info_file: (a file name, nipy default value: dicominfo.txt)
    file to which results are written
    argument: ``--o %s``
sortbyrun: (a boolean)
    assign run numbers
    argument: ``--sortbyrun``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```
dicom_info_file: (an existing file name)
                 text file containing dicom information
```

62.3.15 ReconAll

[Link to code](#)

Wraps the executable 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
```

(continues on next page)

(continued from previous page)

```
>>> reconall_subfields.cmdline
'recon-all -all -i structural.nii -hippocampal-subfields-T2 structural.nii test -
↳subjid foo -sd .'
```

Inputs:

```
[Optional]
mris_make_surfaces: (a unicode string)
    Flags to pass to mris_make_surfaces commands
    mutually_exclusive: expert
mri_segment: (a unicode string)
    Flags to pass to mri_segment commands
    mutually_exclusive: expert
mri_normalize: (a unicode string)
    Flags to pass to mri_normalize commands
    mutually_exclusive: expert
xopts: ('use' or 'clean' or 'overwrite')
    Use, delete or overwrite existing expert options file
    argument: ``-xopts-%s``
hippocampal_subfields_T1: (a boolean)
    segment hippocampal subfields using input T1 scan
    argument: ``-hippocampal-subfields-T1``
talairach: (a unicode string)
    Flags to pass to talairach commands
    mutually_exclusive: expert
mri_aparc2aseg: (a unicode string)
    Flags to pass to mri_aparc2aseg commands
    mutually_exclusive: expert
mris_ca_label: (a unicode string)
    Flags to pass to mris_ca_label commands
    mutually_exclusive: expert
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)
    argument: ``-hippocampal-subfields-T2 %s %s``
mris_sphere: (a unicode string)
    Flags to pass to mris_sphere commands
    mutually_exclusive: expert
use_FLAIR: (a boolean)
    Use FLAIR image to refine the pial surface
    argument: ``-FLAIRpial``
    mutually_exclusive: use_T2
mri_mask: (a unicode string)
    Flags to pass to mri_mask commands
    mutually_exclusive: expert
parallel: (a boolean)
    Enable parallel execution
    argument: ``-parallel``
mris_inflate: (a unicode string)
    Flags to pass to mri_inflate commands
    mutually_exclusive: expert
brainstem: (a boolean)
    Segment brainstem structures
    argument: ``-brainstem-structures``
flags: (a list of items which are a unicode string)
    additional parameters
```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
subject_id: (a unicode string, nipy default value: recon_all)
    subject name
    argument: ``-subjid %s``
mris_smooth: (a unicode string)
    Flags to pass to mris_smooth commands
    mutually_exclusive: expert
mri_ca_label: (a unicode string)
    Flags to pass to mri_ca_label commands
    mutually_exclusive: expert
mris_surf2vol: (a unicode string)
    Flags to pass to mris_surf2vol commands
    mutually_exclusive: expert
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
mris_anatomical_stats: (a unicode string)
    Flags to pass to mris_anatomical_stats commands
    mutually_exclusive: expert
mri_segstats: (a unicode string)
    Flags to pass to mri_segstats commands
    mutually_exclusive: expert
T2_file: (an existing file name)
    Convert T2 image to orig directory
    argument: ``-T2 %s``
mri_em_register: (a unicode string)
    Flags to pass to mri_em_register commands
    mutually_exclusive: expert
mris_fix_topology: (a unicode string)
    Flags to pass to mris_fix_topology commands
    mutually_exclusive: expert
use_T2: (a boolean)
    Use T2 image to refine the pial surface
    argument: ``-T2pial``
    mutually_exclusive: use_FLAIR
mprage: (a boolean)
    Assume scan parameters are MGH MP-RAGE protocol, which produces
    darker gray matter
    argument: ``-mprage``
hires: (a boolean)
    Conform to minimum voxel size (for voxels < 1mm)
    argument: ``-hires``
mri_ca_register: (a unicode string)
    Flags to pass to mri_ca_register commands
    mutually_exclusive: expert
mri_tessellate: (a unicode string)
    Flags to pass to mri_tessellate commands
    mutually_exclusive: expert
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
openmp: (an integer (int or long))
    Number of processors to use in parallel
    argument: ``-openmp %d``
mrisp_paint: (a unicode string)
    Flags to pass to mrisp_paint commands

```

(continues on next page)

(continued from previous page)

```

        mutually_exclusive: expert
big_ventricles: (a boolean)
    For use in subjects with enlarged ventricles
    argument: ``-bigventricles``
mri_edit_wm_with_aseg: (a unicode string)
    Flags to pass to mri_edit_wm_with_aseg commands
    mutually_exclusive: expert
subjects_dir: (an existing directory name)
    path to subjects directory
    argument: ``-sd %s``
FLAIR_file: (an existing file name)
    Convert FLAIR image to orig directory
    argument: ``-FLAIR %s``
mri_remove_neck: (a unicode string)
    Flags to pass to mri_remove_neck commands
    mutually_exclusive: expert
mri_watershed: (a unicode string)
    Flags to pass to mri_watershed commands
    mutually_exclusive: expert
mri_ca_normalize: (a unicode string)
    Flags to pass to mri_ca_normalize commands
    mutually_exclusive: expert
mri_preteess: (a unicode string)
    Flags to pass to mri_preteess commands
    mutually_exclusive: expert
expert: (an existing file name)
    Set parameters using expert file
    argument: ``-expert %s``
hemi: ('lh' or 'rh')
    hemisphere to process
    argument: ``-hemi %s``
mris_register: (a unicode string)
    Flags to pass to mris_register commands
    mutually_exclusive: expert
mri_fill: (a unicode string)
    Flags to pass to mri_fill commands
    mutually_exclusive: expert
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', nipy default value:
    all)
    process directive
    argument: ``-%s``, position: 0
T1_files: (a list of items which are an existing file name)
    name of T1 file to process
    argument: ``-i %s...``

```

Outputs:

```

aseg: (an existing file name)
    Volumetric map of regions from automatic segmentation
volume: (a list of items which are an existing file name)
    Surface maps of cortical volume
label: (a list of items which are an existing file name)
    Volume and surface label files

```

(continues on next page)

(continued from previous page)

```

wmparc_stats: (a list of items which are an existing file name)
    White matter parcellation statistics file
wmparc: (an existing file name)
    Aparc parcellation projected into subcortical white matter
nu: (an existing file name)
    Non-uniformity corrected whole-head volume
sphere: (a list of items which are an existing file name)
    Spherical surface meshes
sulc: (a list of items which are an existing file name)
    Surface maps of sulcal depth
aparc_a2009s_stats: (a list of items which are an existing file name)
    Aparc a2009s parcellation statistics files
T1: (an existing file name)
    Intensity normalized whole-head volume
inflated: (a list of items which are an existing file name)
    Inflated surface meshes
smoothwm: (a list of items which are an existing file name)
    Smoothed original surface meshes
jacobian_white: (a list of items which are an existing file name)
    Distortion required to register to spherical atlas
brain: (an existing file name)
    Intensity normalized brain-only volume
sphere_reg: (a list of items which are an existing file name)
    Spherical registration file
aparc_aseg: (a list of items which are an existing file name)
    Aparc parcellation projected into aseg volume
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
pial: (a list of items which are an existing file name)
    Gray matter/pia mater surface meshes
curv_stats: (a list of items which are an existing file name)
    Curvature statistics files
aseg_stats: (a list of items which are an existing file name)
    Automated segmentation statistics file
curv_pial: (a list of items which are an existing file name)
    Curvature of pial surface
subjects_dir: (an existing directory name)
    Freesurfer subjects directory.
ribbon: (a list of items which are an existing file name)
    Volumetric maps of cortical ribbons
rawavg: (an existing file name)
    Volume formed by averaging input images
avg_curv: (a list of items which are an existing file name)
    Average atlas curvature, sampled to subject
curv: (a list of items which are an existing file name)
    Maps of surface curvature
subject_id: (a unicode string)
    Subject name for whom to retrieve data
white: (a list of items which are an existing file name)
    White/gray matter surface meshes
orig: (an existing file name)
    Base image conformed to Freesurfer space
graymid: (a list of items which are an existing file name)
    Graymid/midthickness surface meshes
entorhinal_exvivo_stats: (a list of items which are an existing file
    name)

```

(continues on next page)

(continued from previous page)

```

    Entorhinal exvivo statistics files
thickness: (a list of items which are an existing file name)
    Surface maps of cortical thickness
BA_stats: (a list of items which are an existing file name)
    Brodmann Area statistics files
annot: (a list of items which are an existing file name)
    Surface annotation files
brainmask: (an existing file name)
    Skull-stripped (brain-only) volume
filled: (an existing file name)
    Subcortical mass volume
aparc_stats: (a list of items which are an existing file name)
    Aparc parcellation statistics files
wm: (an existing file name)
    Segmented white-matter volume
norm: (an existing file name)
    Normalized skull-stripped volume

```

62.3.16 Resample

[Link to code](#)

Wraps the executable 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
    argument: ``-i %s``, position: -2
voxel_size: (a tuple of the form: (a float, a float, a float))
    triplet of output voxel sizes
    argument: ``-vs %.2f %.2f %.2f``

[Optional]
subjects_dir: (an existing directory name)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
resampled_file: (a file name)
    output filename
    argument: ``-o %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

```

(continues on next page)

(continued from previous page)

```
value of class 'str', nipy default value: {}  
Environment variables
```

Outputs:

```
resampled_file: (an existing file name)  
output filename
```

62.3.17 RobustRegister[Link to code](#)Wraps the executable 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  
    argument: ``--satit``  
    mutually_exclusive: outlier_sens  
target_file: (an existing file name)  
    target volume for the registration  
    argument: ``--dst %s``  
outlier_sens: (a float)  
    set outlier sensitivity explicitly  
    argument: ``--sat %.4f``  
    mutually_exclusive: auto_sens  
source_file: (an existing file name)  
    volume to be registered  
    argument: ``--mov %s``  
  
[Optional]  
no_init: (a boolean)  
    skip transform init  
    argument: ``--noinit``  
out_reg_file: (a bool or None or a file name, nipy default value:  
    True)  
    registration file; either True or filename
```

(continues on next page)

(continued from previous page)

```

        argument: ``--lta %s``
trans_only: (a boolean)
    find 3 parameter translation only
    argument: ``--transonly``
outlier_limit: (a float)
    set maximal outlier limit in satit
    argument: ``--wlimit %.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
high_iterations: (an integer (int or long))
    max # of times on highest resolution
    argument: ``--highit %d``
no_multi: (a boolean)
    work on highest resolution
    argument: ``--nomulti``
half_source_xfm: (a boolean or a file name)
    write transform from source to halfway space
    argument: ``--halfmovlta %s``
half_weights: (a boolean or a file name)
    write weights volume mapped to halfway space
    argument: ``--halfweights %s``
mask_target: (an existing file name)
    image to mask target volume with
    argument: ``--maskdst %s``
least_squares: (a boolean)
    use least squares instead of robust estimator
    argument: ``--leastquares``
subjects_dir: (an existing directory name)
    subjects directory
in_xfm_file: (an existing file name)
    use initial transform on source
    argument: ``--transform``
mask_source: (an existing file name)
    image to mask source volume with
    argument: ``--maskmov %s``
force_double: (a boolean)
    use double-precision intensities
    argument: ``--doubleprec``
subsample_thresh: (an integer (int or long))
    subsample if dimension is above threshold size
    argument: ``--subsample %d``
registered_file: (a boolean or a file name)
    registered image; either True or filename
    argument: ``--warp %s``
half_targ_xfm: (a boolean or a file name)
    write transform from target to halfway space
    argument: ``--halfdstlta %s``
half_source: (a boolean or a file name)
    write source volume mapped to halfway space
    argument: ``--halfmov %s``
weights_file: (a boolean or a file name)
    weights image to write; either True or filename
    argument: ``--weights %s``
init_orient: (a boolean)
    use moments for initial orient (recommended for stripped brains)

```

(continues on next page)

(continued from previous page)

```

        argument: ``--initorient``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
max_iterations: (an integer (int or long))
        maximum # of times on each resolution
        argument: ``--maxit %d``
est_int_scale: (a boolean)
        estimate intensity scale (recommended for unnormalized images)
        argument: ``--iscale``
write_vo2vox: (a boolean)
        output vox2vox matrix (default is RAS2RAS)
        argument: ``--vox2vox``
force_float: (a boolean)
        use float intensities
        argument: ``--floattype``
iteration_thresh: (a float)
        stop iterations when below threshold
        argument: ``--epsit %.3f``
half_targ: (a boolean or a file name)
        write target volume mapped to halfway space
        argument: ``--halfdst %s``

```

Outputs:

```

registered_file: (an existing file name)
        output image with registration applied
half_weights: (an existing file name)
        weights image mapped to halfway space
half_source_xfm: (an existing file name)
        transform file to map source image to halfway space
half_targ: (an existing file name)
        target image mapped to halfway space
out_reg_file: (an existing file name)
        output registration file
weights_file: (an existing file name)
        image of weights used
half_targ_xfm: (an existing file name)
        transform file to map target image to halfway space
half_source: (an existing file name)
        source image mapped to halfway space

```

62.3.18 SegmentCC[Link to code](#)Wraps the executable 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]
subject_id: (a string, nipy default value: subject_id)
    Subject name
    argument: ``%s``, position: -1
in_file: (an existing file name)
    Input aseg file to read from subjects directory
    argument: ``-aseg %s``
out_rotation: (a file name)
    Global filepath for writing rotation lta
    argument: ``-lta %s``
in_norm: (an existing file name)
    Required undocumented input {subject}/mri/norm.mgz

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
out_file: (a file name)
    Filename to write aseg including CC
    argument: ``-o %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
subjects_dir: (an existing directory name)
    subjects directory
```

Outputs:

```
out_file: (a file name)
    Output segmentation uncluding corpus collosum
out_rotation: (a file name)
    Output lta rotation file
```

62.3.19 SegmentWM

[Link to code](#)

Wraps the executable 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
         argument: ``%s``, position: -2
out_file: (a file name)
         File to be written as output for SegmentWM
         argument: ``%s``, position: -1

[Optional]
subjects_dir: (an existing directory name)
              subjects directory
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
out_file: (a file name)
          Output white matter segmentation
```

62.3.20 Smooth

[Link to code](#)

Wraps the executable command `mrisc_volsmooth`.

Use FreeSurfer `mrisc_volsmooth` to smooth a volume

This function smoothes cortical regions on a surface and non-cortical regions in volume.

Note: Cortical voxels are mapped to the surface (3D->2D) and then the smoothed values from the surface are put back into the volume to fill the cortical ribbon. If data is smoothed with this algorithm, one has to be careful about how further processing is interpreted.

Examples

```
>>> from nipy.interfaces.freesurfer import Smooth
>>> smoothvol = Smooth(in_file='functional.nii', smoothed_file = 'foo_out.nii',
↳ reg_file='register.dat', surface_fwhm=10, vol_fwhm=6)
>>> smoothvol.cmdline
'mrisc_volsmooth --i functional.nii --reg register.dat --o foo_out.nii --fwhm 10.
↳ 000000 --vol-fwhm 6.000000'
```

Inputs:

```

[Mandatory]
reg_file: (an existing file name)
    registers volume to surface anatomical
    argument: ``--reg %s``
surface_fwhm: (a floating point number >= 0.0)
    surface FWHM in mm
    argument: ``--fwhm %f``
    mutually_exclusive: num_iters
    requires: reg_file
in_file: (an existing file name)
    source volume
    argument: ``--i %s``
num_iters: (a long integer >= 1)
    number of iterations instead of fwhm
    argument: ``--nitters %d``
    mutually_exclusive: surface_fwhm

[Optional]
vol_fwhm: (a floating point number >= 0.0)
    volume smoothing outside of surface
    argument: ``--vol-fwhm %f``
subjects_dir: (an existing directory name)
    subjects directory
smoothed_file: (a file name)
    output volume
    argument: ``--o %s``
proj_frac_avg: (a tuple of the form: (a float, a float, a float))
    average a long normal min max delta
    argument: ``--projfrac-avg %.2f %.2f %.2f``
    mutually_exclusive: proj_frac
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
proj_frac: (a float)
    project frac of thickness a long surface normal
    argument: ``--projfrac %s``
    mutually_exclusive: proj_frac_avg

```

Outputs:

```

smoothed_file: (an existing file name)
    smoothed input volume

```

62.3.21 SynthesizeFLASH

[Link to code](#)

Wraps the executable command `mri_synthesize`.

Synthesize a FLASH acquisition from T1 and proton density maps.

Examples


```
>>> from nipyte.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]
pd_image: (an existing file name)
    image of proton density values
    argument: ``%s``, position: 6
tr: (a float)
    repetition time (in msec)
    argument: ``%.2f``, position: 2
flip_angle: (a float)
    flip angle (in degrees)
    argument: ``%.2f``, position: 3
te: (a float)
    echo time (in msec)
    argument: ``%.3f``, position: 4
t1_image: (an existing file name)
    image of T1 values
    argument: ``%s``, position: 5

[Optional]
out_file: (a file name)
    image to write
    argument: ``%s``
subjects_dir: (an existing directory name)
    subjects directory
fixed_weighting: (a boolean)
    use a fixed weighting to generate optimal gray/white contrast
    argument: ``-w``, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a 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
```

Outputs:

```
out_file: (an existing file name)
    synthesized FLASH acquisition
```

62.3.22 UnpackSDICOMDir

[Link to code](#)

Wraps the executable command `unpacksdcmdir`.

Use `unpacksdcmdir` to convert dicom files

Call `unpacksdcmdir -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
'unpacksdcmdir -generic -targ . -run 5 mprage nii struct -src .'
```

Inputs:

```
[Mandatory]
config: (an existing file name)
    specify unpacking rules in file
    argument: ``-cfg %s``
    mutually_exclusive: run_info, config, seq_config
source_dir: (an existing directory name)
    directory with the DICOM files
    argument: ``-src %s``
run_info: (a tuple of the form: (an integer (int or long), a unicode
    string, a unicode string, a unicode string))
    runno subdir format name : spec unpacking rules on cmdline
    argument: ``-run %d %s %s %s``
    mutually_exclusive: run_info, config, seq_config
seq_config: (an existing file name)
    specify unpacking rules based on sequence
    argument: ``-seqcfg %s``
    mutually_exclusive: run_info, config, seq_config

[Optional]
subjects_dir: (an existing directory name)
    subjects directory
no_info_dump: (a boolean)
    do not create infodump file
    argument: ``-noinfodump``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
output_dir: (a directory name)
    top directory into which the files will be unpacked
    argument: ``-targ %s``
dir_structure: ('fsfast' or 'generic')
    unpack to specified directory structures
    argument: ``-%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
scan_only: (an existing file name)
    only scan the directory and put result in file
    argument: ``-scanonly %s``
spm_zeropad: (an integer (int or long))
    set frame number zero padding width for SPM
    argument: ``-nspmzeropad %d``
no_unpack_err: (a boolean)
    do not try to unpack runs with errors
```

(continues on next page)

(continued from previous page)

```

        argument: ``-no-unpackerr``
log_file: (an existing file name)
        explicilty set log file
        argument: ``-log %s``

```

Outputs:

None

62.3.23 WatershedSkullStrip

[Link to code](#)

Wraps the executable 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 dertermine 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
        argument: ``%s``, position: -2
out_file: (a file name, nipy default value: brainmask.auto.mgz)
        output volume
        argument: ``%s``, position: -1

[Optional]
subjects_dir: (an existing directory name)
        subjects directory
brain_atlas: (an existing file name)
        argument: ``-brain_atlas %s``, position: -4
transform: (a file name)
        undocumented
        argument: ``%s``, position: -3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a

```

(continues on next page)

(continued from previous page)

```

        value of class 'str', nipy default value: {})
    Environment variables
t1: (a boolean)
    specify T1 input volume (T1 grey value = 110)
    argument: ``-T1``

```

Outputs:

```

out_file: (a file name)
    skull stripped brain volume

```

62.4 interfaces.freesurfer.registration

62.4.1 EMRegister

[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -3
template: (an existing file name)
    template gca
    argument: ``%s``, position: -2

[Optional]
mask: (an existing file name)
    use volume as a mask
    argument: ``-mask %s``
out_file: (a file name)
    output transform
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
transform: (an existing file name)
    Previously computed transform
    argument: ``-t %s``
skull: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        align to atlas containing skull (uns=5)
        argument: ``-skull``
num_threads: (an integer (int or long))
        allows for specifying more threads
nbrspacing: (an integer (int or long))
        align to atlas containing skull setting unknown_nbr_spacing =
        nbrspacing
        argument: ``-uns %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
subjects_dir: (an existing directory name)
        subjects directory

```

Outputs:

```

out_file: (a file name)
        output transform

```

62.4.2 MPRtoMNI305[Link to code](#)Wraps the executable command `mpr2mni305`.For complete details, see [FreeSurfer documentation](#)**Examples**

```

>>> from nipype.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, nipype default value: )
        TODO
target: (a string, nipype default value: )
        input atlas file

[Optional]
subjects_dir: (an existing directory name)
        subjects directory
args: (a unicode string)
        Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
in_file: (a file name, nipy default value: )
        the input file prefix for MPRtoMNI305
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

out_file: (a file name)
        The output file '<in_file>_to_<target>_t4_vox2vox.txt'
log_file: (an existing file name, nipy default value:
        output.nipy)
        The output log

```

62.4.3 MRICoreg

[Link to code](#)Wraps the executable command `mri_coreg`.

This program registers one volume to another

`mri_coreg` is a C reimplementation of `spm_coreg` in FreeSurfer**Examples**

```

>>> from nipy.interfaces.freesurfer import MRICoreg
>>> coreg = MRICoreg()
>>> coreg.inputs.source_file = 'moving1.nii'
>>> coreg.inputs.reference_file = 'fixed1.nii'
>>> coreg.inputs.subjects_dir = '.'
>>> coreg.cmdline
'mri_coreg --lta ../registration.lta --ref fixed1.nii --mov moving1.nii --sd .'

```

If passing a subject ID, the reference mask may be disabled:

```

>>> coreg = MRICoreg()
>>> coreg.inputs.source_file = 'moving1.nii'
>>> coreg.inputs.subjects_dir = '.'
>>> coreg.inputs.subject_id = 'fsaverage'
>>> coreg.inputs.reference_mask = False
>>> coreg.cmdline
'mri_coreg --s fsaverage --no-ref-mask --lta ../registration.lta --mov moving1.
↪nii --sd .'

```

Spatial scales may be specified as a list of one or two separations:

```

>>> coreg.inputs.sep = [4]
>>> coreg.cmdline
'mri_coreg --s fsaverage --no-ref-mask --lta ../registration.lta --sep 4 --mov_
↪moving1.nii --sd .'

```

```

>>> coreg.inputs.sep = [4, 5]
>>> coreg.cmdline
'mri_coreg --s fsaverage --no-ref-mask --lta ../registration.lta --sep 4 --sep 5_
↪--mov moving1.nii --sd .'

```

Inputs:

```

[Mandatory]
subject_id: (a unicode string)
    freesurfer subject ID (implies ``reference_mask == aparc+aseg.mgz``
    unless otherwise specified)
    argument: ``--s %s``, position: 1
    mutually_exclusive: reference_file
    requires: subjects_dir
reference_file: (a file name)
    reference (target) file
    argument: ``--ref %s``
    mutually_exclusive: subject_id
source_file: (a file name)
    source file to be registered
    argument: ``--mov %s``

[Optional]
linmintol: (a float)
    argument: ``--linmintol %e``
saturation_threshold: (0.0 <= a floating point number <= 100.0)
    saturation threshold (default=9.999)
    argument: ``--sat %g``
no_smooth: (a boolean)
    do not apply smoothing to either reference or source file
    argument: ``--no-smooth``
brute_force_samples: (an integer (int or long))
    number of samples in brute force search
    argument: ``--bf-nsamp %d``
    mutually_exclusive: no_brute_force
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
source_oob: (a boolean)
    count source voxels that are out-of-bounds as 0
    argument: ``--mov-oob``
ref_fwhm: (a float)
    apply smoothing to reference file
    argument: ``--ref-fwhm``
no_cras0: (a boolean)
    do not set translation parameters to align centers of source and
    reference files
    argument: ``--no-cras0``
initial_scale: (a tuple of the form: (a float, a float, a float))
    initial scale
    argument: ``--scale %g %g %g``
dof: (6 or 9 or 12)
    number of transform degrees of freedom
    argument: ``--dof %d``
out_params_file: (a bool or None or a file name)
    output parameters file
    argument: ``--params %s``
out_reg_file: (a bool or None or a file name)
    output registration file (REG format)
    argument: ``--regdat %s``
sep: (a list of from 1 to 2 items which are any value)
    set spatial scales, in voxels (default [2, 4])

```

(continues on next page)

(continued from previous page)

```

        argument: ``--sep %s...``
conform_reference: (a boolean)
    conform reference without rescaling
    argument: ``--conf-ref``
subjects_dir: (an existing directory name)
    FreeSurfer SUBJECTS_DIR
    argument: ``--sd %s``
ftol: (a float)
    floating-point tolerance (default=1e-7)
    argument: ``--ftol %e``
no_coord_dithering: (a boolean)
    turn off coordinate dithering
    argument: ``--no-coord-dither``
brute_force_limit: (a float)
    constrain brute force search to +/- lim
    argument: ``--bf-lim %g``
    mutually_exclusive: no_brute_force
initial_rotation: (a tuple of the form: (a float, a float, a float))
    initial rotation in degrees
    argument: ``--rot %g %g %g``
out_lta_file: (a bool or None or a file name, nipy default value:
    True)
    output registration file (LTA format)
    argument: ``--lta %s``
no_intensity_dithering: (a boolean)
    turn off intensity dithering
    argument: ``--no-intensity-dither``
max_iters: (a long integer >= 1)
    maximum iterations (default: 4)
    argument: ``--nittersmax %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
no_brute_force: (a boolean)
    do not brute force search
    argument: ``--no-bf``
initial_shear: (a tuple of the form: (a float, a float, a float))
    initial shear (Hxy, Hxz, Hyz)
    argument: ``--shear %g %g %g``
reference_mask: (a bool or None or a unicode string)
    mask reference volume with given mask, or None if ``False``
    argument: ``--ref-mask %s``, position: 2
num_threads: (an integer (int or long))
    number of OpenMP threads
    argument: ``--threads %d``
initial_translation: (a tuple of the form: (a float, a float, a
    float))
    initial translation in mm (implies no_cras0)
    argument: ``--trans %g %g %g``
source_mask: (a unicode string)
    mask source file with given mask
    argument: ``--mov-mask``

```

Outputs:

```

out_params_file: (an existing file name)
    output parameters file

```

(continues on next page)

(continued from previous page)

```

out_reg_file: (an existing file name)
               output registration file
out_lta_file: (an existing file name)
               output LTA-style registration file

```

62.4.4 Paint

[Link to code](#)

Wraps the executable command `mrisp_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 nipy.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
'mrisp_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'
          argument: ``%s``, position: -2
template: (an existing file name)
          Template file
          argument: ``%s``, position: -3

[Optional]
out_file: (a file name)
          File containing a surface-worth of per-vertex values, saved in
          'curvature' format.
          argument: ``%s``, position: -1
averages: (an integer (int or long))
          Average curvature patterns
          argument: ``-a %d``
subjects_dir: (an existing directory name)
              subjects directory
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
template_param: (an integer (int or long))
                Frame number of the input template

```

Outputs:

```
out_file: (a file name)
    File containing a surface-worth of per-vertex values, saved in
    'curvature' format.
```

62.4.5 Register

[Link to code](#)

Wraps the executable command `mris_register`.

This program registers a surface to an average surface template.

Examples

```
>>> from nipyype.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
'mris_register -curv lh.pial aseg.mgz lh.pial.reg'
```

Inputs:

```
[Mandatory]
in_surf: (an existing file name)
    Surface to register, often {hemi}.sphere
    argument: ``%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.
    argument: ``%s``, position: -2
in_sulc: (an existing file name)
    Undocumented mandatory input file
    ${SUBJECTS_DIR}/surf/{hemisphere}.sulc

[Optional]
subjects_dir: (an existing directory name)
    subjects directory
out_file: (a file name)
    Output surface file to capture registration
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
curv: (a boolean)
    Use smoothwm curvature for final alignment
    argument: ``-curv``
    requires: in_smoothwm
in_smoothwm: (an existing file name)
    Undocumented input file ${SUBJECTS_DIR}/surf/{hemisphere}.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', nipyype default value: {})
    Environment variables
```

Outputs:

```
out_file: (a file name)
          Output surface file to capture registration
```

62.4.6 RegisterAVItoTalairach[Link to code](#)

Wraps the executable 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 RegisterAVItoTalairach
>>> register = RegisterAVItoTalairach()
>>> 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
         argument: ``%s``, position: 0
target: (an existing file name)
         The target file
         argument: ``%s``, position: 1
vox2vox: (an existing file name)
         The vox2vox file
         argument: ``%s``, position: 2

[Optional]
out_file: (a file name, nipy default value: talairach.auto.xfm)
          The transform output
          argument: ``%s``, position: 3
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

(continues on next page)

(continued from previous page)

```
subjects_dir: (an existing directory name)
              subjects directory
```

Outputs:

```
out_file: (a file name)
           The output file for RegisterAVitoTalairach
log_file: (an existing file name, nipype default value:
           output.nipype)
           The output log
```

62.5 interfaces.freesurfer.utils

62.5.1 AddXFormToHeader

Link to code

Wraps the executable 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 nipype.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]
transform: (a file name)
           xfm file
           argument: ``%s``, position: -3
in_file: (an existing file name)
          input volume
          argument: ``%s``, position: -2

[Optional]
copy_name: (a boolean)
           do not try to load the xfmfile, just copy name
           argument: ``-c``
out_file: (a file name, nipype default value: output.mgz)
          output volume
          argument: ``%s``, position: -1
```

(continues on next page)

(continued from previous page)

```

verbose: (a boolean)
    be verbose
    argument: ``-v``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a 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
subjects_dir: (an existing directory name)
    subjects directory

```

Outputs:

```

out_file: (an existing file name)
    output volume

```

62.5.2 Aparc2Aseg

[Link to code](#)

Wraps the executable 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 nipyype.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]
subject_id: (a string, nipyype default value: subject_id)

```

(continues on next page)

(continued from previous page)

```

    Subject being processed
    argument: ``--s %s``
lh_pial: (an existing file name)
    Input file must be <subject_id>/surf/lh.pial
rh_pial: (an existing file name)
    Input file must be <subject_id>/surf/rh.pial
out_file: (a file name)
    Full path of file to save the output segmentation in
    argument: ``--o %s``
lh_ribbon: (an existing file name)
    Input file must be <subject_id>/mri/lh.ribbon.mgz
rh_ribbon: (an existing file name)
    Input file must be <subject_id>/mri/rh.ribbon.mgz
lh_white: (an existing file name)
    Input file must be <subject_id>/surf/lh.white
lh_annotation: (an existing file name)
    Input file must be <subject_id>/label/lh.aparc.annot
rh_annotation: (an existing file name)
    Input file must be <subject_id>/label/rh.aparc.annot
ribbon: (an existing file name)
    Input file must be <subject_id>/mri/ribbon.mgz
rh_white: (an existing file name)
    Input file must be <subject_id>/surf/rh.white

[Optional]
aseg: (an existing file name)
    Input aseg file
    argument: ``--aseg %s``
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
    argument: ``--labelwm``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
volmask: (a boolean)
    Volume mask flag
    argument: ``--volmask``
subjects_dir: (an existing directory name)
    subjects directory
a2009s: (a boolean)
    Using the a2009s atlas
    argument: ``--a2009s``
hypo_wm: (a boolean)
    Label hypointensities as WM
    argument: ``--hypo-as-wm``
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the input
    files to the node directory.
rip_unknown: (a boolean)
    Do not label WM based on 'unknown' corical label
    argument: ``--rip-unknown``
ctxseg: (an existing file name)
    argument: ``--ctxseg %s``
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

Additional parameters to the command
argument: ``%s``
filled: (an existing file name)
Implicit input filled file. Only required with FS v5.3.

```

Outputs:

```

out_file: (a file name)
Output aseg file
argument: ``%s``

```

62.5.3 Apas2Aseg[Link to code](#)

Wraps the executable 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
argument: ``--i %s``
out_file: (a file name)
Output aseg file
argument: ``--o %s``

[Optional]
subjects_dir: (an existing directory name)
subjects directory
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipy default value: {})
Environment variables

```

Outputs:

```

out_file: (a file name)
Output aseg file
argument: ``%s``

```

62.5.4 ApplyMask

[Link to code](#)

Wraps the executable 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)
        argument: ``%s``, position: -3
mask_file: (an existing file name)
        image defining mask space
        argument: ``%s``, position: -2

[Optional]
subjects_dir: (an existing directory name)
              subjects directory
out_file: (a file name)
          final image to write
          argument: ``%s``, position: -1
xfm_source: (an existing file name)
            image defining transform source space
            argument: ``-lta_src %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
xfm_file: (an existing file name)
          LTA-format transformation matrix to align mask with input
          argument: ``-xform %s``
mask_thresh: (a float)
             threshold mask before applying
             argument: ``-T %.4f``
invert_xfm: (a boolean)
            invert transformation
            argument: ``-invert``
use_abs: (a boolean)
         take absolute value of mask before applying
         argument: ``-abs``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
transfer: (an integer (int or long))
          transfer only voxel value # from mask to out
          argument: ``-transfer %d``
xfm_target: (an existing file name)
            image defining transform target space
            argument: ``-lta_dst %s``
keep_mask_deletion_edits: (a boolean)
                          transfer voxel-deletion edits (voxels=1) from mask to out vol
                          argument: ``-keep_mask_deletion_edits``
```

Outputs:

```
out_file: (an existing file name)
          masked image
```


62.5.5 CheckTalairachAlignment

[Link to code](#)

Wraps the executable 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
    argument: ``-xfm %s``, position: -1
    mutually_exclusive: subject
subject: (a string)
    specify subject's name
    argument: ``-subj %s``, position: -1
    mutually_exclusive: in_file

[Optional]
subjects_dir: (an existing directory name)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
threshold: (a float, nipy default value: 0.01)
    Talairach transforms for subjects with p-values <= T are considered
    as very unlikely default=0.010
    argument: ``-T %.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
```

Outputs:

```
out_file: (a file name)
    The input file for CheckTalairachAlignment
```

62.5.6 Contrast

[Link to code](#)

Wraps the executable command `pctsurfcon`.

Compute surface-wise gray/white contrast

Examples

```
>>> from nipyne.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
subject_id: (a string, nipyne default value: subject_id)
    Subject being processed
    argument: `--s %s`
orig: (an existing file name)
    Implicit input file mri/orig.mgz
white: (an existing file name)
    Input file must be <subject_id>/surf/<hemisphere>.white
cortex: (a file name)
    Input cortex label must be
    <subject_id>/label/<hemisphere>.cortex.label
thickness: (an existing file name)
    Input file must be <subject_id>/surf/?h.thickness
rawavg: (an existing file name)
    Implicit input file mri/rawavg.mgz
hemisphere: ('lh' or 'rh')
    Hemisphere being processed
    argument: `--%s-only`

[Optional]
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', nipyne default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: `-%s`
subjects_dir: (an existing directory name)
    subjects directory
```

Outputs:

```
out_stats: (a file name)
    Output stats file from Contrast
out_contrast: (a file name)
    Output contrast file from Contrast
```

(continues on next page)

(continued from previous page)

```
out_log: (an existing file name)
        Output log from Contrast
```

62.5.7 Curvature

[Link to code](#)

Wraps the executable command `mris_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
        argument: ``%s``, position: -2

[Optional]
distances: (a tuple of the form: (an integer (int or long), an
        integer (int or long)))
        Undocumented input integer distances
        argument: ``-distances %d %d``
averages: (an integer (int or long))
        Perform this number iterative averages of curvature measure before
        saving
        argument: ``-a %d``
subjects_dir: (an existing directory name)
        subjects directory
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
save: (a boolean)
        Save curvature files (will only generate screen output without this
        option)
        argument: ``-w``
n: (a boolean)
        Undocumented boolean flag
        argument: ``-n``
threshold: (a float)
        Undocumented input threshold
        argument: ``-thresh %.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
copy_input: (a boolean)
        Copy input file to current directory
```

Outputs:

```

out_mean: (a file name)
           Mean curvature output file
out_gauss: (a file name)
           Gaussian curvature output file

```

62.5.8 CurvatureStats[Link to code](#)

Wraps the executable command `mris_curvature_stats`.

In its simplest usage, 'mris_curvature_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, 'mris_curvature_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]
subject_id: (a string, nipy default value: subject_id)
            Subject being processed
            argument: ``%s``, position: -4
curvfile1: (an existing file name)
            Input file for CurvatureStats
            argument: ``%s``, position: -2
hemisphere: ('lh' or 'rh')
            Hemisphere being processed
            argument: ``%s``, position: -3
curvfile2: (an existing file name)
            Input file for CurvatureStats
            argument: ``%s``, position: -1

[Optional]
subjects_dir: (an existing directory name)

```

(continues on next page)

(continued from previous page)

```

    subjects directory
copy_inputs: (a boolean)
    If running as a node, set this to True.This will copy the input
    files to the node directory.
surface: (an existing file name)
    Specify surface file for CurvatureStats
    argument: ``-F %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
write: (a boolean)
    Write curvature files
    argument: ``--writeCurvatureFiles``
out_file: (a file name)
    Output curvature stats file
    argument: ``-o %s``
values: (a boolean)
    Triggers a series of derived curvature values
    argument: ``-G``
min_max: (a boolean)
    Output min / max information for the processed curvature.
    argument: ``-m``

```

Outputs:

```

out_file: (a file name)
    Output curvature stats file

```

62.5.9 EulerNumber[Link to code](#)Wraps the executable command `mrisc EulerNumber`.

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
'mrisc EulerNumber lh.pial'

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
    Input file for EulerNumber
    argument: ``%s``, position: -1

[Optional]
subjects_dir: (an existing directory name)
    subjects directory

```

(continues on next page)

(continued from previous page)

```
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a 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
```

Outputs:

```
out_file: (a file name)
          Output file for EulerNumber
```

62.5.10 ExtractMainComponent

[Link to code](#)

Wraps the executable command `mris_extract_main_component`.

Extract the main component of a tessellated surface

Examples

```
>>> from nipyne.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
        argument: ``%s``, position: 1

[Optional]
out_file: (a file name)
          surface containing main component
          argument: ``%s``, position: 2
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a 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
```

Outputs:

```
out_file: (an existing file name)
          surface containing main component
```

62.5.11 FixTopology

[Link to code](#)

Wraps the executable command `mris_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]
in_orig: (an existing file name)
    Undocumented input file <hemisphere>.orig
subject_id: (a string, nipy default value: subject_id)
    Subject being processed
    argument: ``%s``, position: -2
in_brain: (an existing file name)
    Implicit input brain.mgz
in_inflated: (an existing file name)
    Undocumented input file <hemisphere>.inflated
hemisphere: (a string)
    Hemisphere being processed
    argument: ``%s``, position: -1
copy_inputs: (a boolean)
    If running as a node, set this to True otherwise, the topology
    fixing will be done in place.
in_wm: (an existing file name)
    Implicit input wm.mgz

[Optional]
subjects_dir: (an existing directory name)
    subjects directory
ga: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    argument: ``-ga``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
sphere: (a file name)
    Sphere input file
    argument: ``-sphere %s``
seed: (an integer (int or long))
    Seed for setting random number generator
    argument: ``-seed %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
mgz: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
```

(continues on next page)

(continued from previous page)

```
argument: ``-mgz``
```

Outputs:

```
out_file: (a file name)
          Output file for FixTopology
```

62.5.12 Jacobian

[Link to code](#)

Wraps the executable command `mris_jacobian`.

This program computes the Jacobian of a surface mapping.

Examples

```
>>> from nipyne.interfaces.freesurfer import Jacobian
>>> jacobian = Jacobian()
>>> jacobian.inputs.in_origsurf = 'lh.pial'
>>> jacobian.inputs.in_mappedsurf = 'lh.pial'
>>> jacobian.cmdline
'mris_jacobian lh.pial lh.pial lh.jacobian'
```

Inputs:

```
[Mandatory]
in_origsurf: (an existing file name)
              Original surface
              argument: ``%s``, position: -3
in_mappedsurf: (an existing file name)
                Mapped surface
                argument: ``%s``, position: -2

[Optional]
out_file: (a file name)
           Output Jacobian of the surface mapping
           argument: ``%s``, position: -1
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
environ: (a dictionary with keys which are a 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
subjects_dir: (an existing directory name)
              subjects directory
```

Outputs:

```
out_file: (a file name)
          Output Jacobian of the surface mapping
```

62.5.13 LTAConvert

[Link to code](#)

Wraps the executable command `lta_convert`.

Convert different transformation formats. Some formats may require you to pass an image if the geometry information is missing from the transform file format.

For complete details, see the [lta_convert](#) documentation.

Inputs:

```
[Mandatory]
in_fsl: (an existing file name)
        input transform of FSL type
        argument: ``--infs1 %s``
        mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg,
                             in_itk
in_mni: (an existing file name)
        input transform of MNI/XFM type
        argument: ``--inmni %s``
        mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg,
                             in_itk
in_niftyreg: (an existing file name)
        input transform of Nifty Reg type (inverse RAS2RAS)
        argument: ``--inniftyreg %s``
        mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg,
                             in_itk
in_itk: (an existing file name)
        input transform of ITK type
        argument: ``--initk %s``
        mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg,
                             in_itk
in_lta: (an existing file name or 'identity.noimage')
        input transform of LTA type
        argument: ``--inlta %s``
        mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg,
                             in_itk
in_reg: (an existing file name)
        input transform of TK REG type (deprecated format)
        argument: ``--inreg %s``
        mutually_exclusive: in_lta, in_fsl, in_mni, in_reg, in_niftyreg,
                             in_itk

[Optional]
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
out_itk: (a boolean or a file name)
        output transform in ITK format
        argument: ``--outitk %s``
ltavox2vox: (a boolean)
        argument: ``--ltavox2vox``
        requires: out_lta
out_reg: (a boolean or a file name)
        output transform in reg dat format
        argument: ``--outreg %s``
out_fsl: (a boolean or a file name)
        output transform in FSL format
        argument: ``--outfsl %s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
source_file: (an existing file name)
        argument: ``--src %s``
target_file: (an existing file name)
```

(continues on next page)

(continued from previous page)

```

        argument: ``--trg %s``
invert: (a boolean)
        argument: ``--invert``
out_lta: (a boolean or a file name)
        output linear transform (LTA Freesurfer format)
        argument: ``--outlta %s``
target_conform: (a boolean)
        argument: ``--trgconform``
out_mni: (a boolean or a file name)
        output transform in MNI/XFM format
        argument: ``--outmni %s``

```

Outputs:

```

out_mni: (an existing file name)
        output transform in MNI/XFM format
out_fsl: (an existing file name)
        output transform in FSL format
out_reg: (an existing file name)
        output transform in reg dat format
out_lta: (an existing file name)
        output linear transform (LTA Freesurfer format)
out_itk: (an existing file name)
        output transform in ITK format

```

62.5.14 MRIFill[Link to code](#)Wraps the executable 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
        argument: ``%s``, position: -2
out_file: (a file name)
        Output filled volume file name for MRIFill
        argument: ``%s``, position: -1

[Optional]
subjects_dir: (an existing directory name)
        subjects directory
args: (a unicode string)
        Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
transform: (an existing file name)
        Input transform file for MRIFill
        argument: ``-xform %s``
log_file: (a file name)
        Output log file for MRIFill
        argument: ``-a %s``
segmentation: (an existing file name)
        Input segmentation file for MRIFill
        argument: ``-segmentation %s``
environ: (a dictionary with keys which are a 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

```

Outputs:

```

out_file: (a file name)
        Output file from MRIFill
log_file: (a file name)
        Output log file from MRIFill

```

62.5.15 MRIMarchingCubes[Link to code](#)Wraps the executable command `mri_mc`.Uses Freesurfer's `mri_mc` to create surfaces by tessellating a given input volume**Example**

```

>>> import nipype.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.
        argument: ``%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)
        argument: ``%d``, position: 2

[Optional]
connectivity_value: (an integer (int or long), nipype default value:
        1)
        Alter the marching cubes connectivity: 1=6+,2=18,3=6,4=26
        (default=1)
        argument: ``%d``, position: -1
out_file: (a file name)
        output filename or True to generate one

```

(continues on next page)

(continued from previous page)

```

        argument: ``./%s``, position: -2
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
    Environment variables
subjects_dir: (an existing directory name)
    subjects directory

```

Outputs:

```

surface: (an existing file name)
    binary surface of the tessellation

```

62.5.16 MRIPretess[Link to code](#)Wraps the executable command `mri_pretess`.Uses Freesurfer's `mri_pretess` to prepare volumes to be tessellated.**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_pretess -nocorners wm.mgz wm norm.mgz wm_pretesswm.mgz'
>>> pretess.run()

```

Inputs:

```

[Mandatory]
in_filled: (an existing file name)
    filled volume, usually wm.mgz
    argument: ``%s``, position: -4
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)
    argument: ``%s``, position: -3
in_norm: (an existing file name)
    the normalized, brain-extracted T1w image. Usually norm.mgz
    argument: ``%s``, position: -2

[Optional]
nocorners: (a boolean)
    do not remove corner configurations in addition to edge ones.

```

(continues on next page)

(continued from previous page)

```

        argument: ``-nocorners``
test: (a boolean)
    adds a voxel that should be removed by mri_pretess. 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.
    argument: ``-test``
subjects_dir: (an existing directory name)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
keep: (a boolean)
    keep WM edits
    argument: ``-keep``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
out_file: (a file name)
    the output file after mri_pretess.
    argument: ``%s``, position: -1

```

Outputs:

```

out_file: (an existing file name)
    output file after mri_pretess

```

62.5.17 MRITessellate[Link to code](#)Wraps the executable 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.MRITessellate()
>>> 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.
    argument: ``%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)
    argument: ``%d``, position: -2

[Optional]
use_real_RAS_coordinates: (a boolean)
    Saves surface with real RAS coordinates where c_(r,a,s) != 0

```

(continues on next page)

(continued from previous page)

```

        argument: ``-n``
tessellate_all_voxels: (a boolean)
    Tessellate the surface of all voxels with different labels
    argument: ``-a``
out_file: (a file name)
    output filename or True to generate one
    argument: ``%s``, position: -1
subjects_dir: (an existing directory name)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

surface: (an existing file name)
    binary surface of the tessellation

```

62.5.18 MRIsCalc[Link to code](#)

Wraps the executable 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)
    argument: ``%s``, position: -2
in_file1: (an existing file name)
    Input file 1
    argument: ``%s``, position: -3
out_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

        Output file after calculation
        argument: ``-o %s``

[Optional]
in_int: (an integer (int or long))
        Input integer
        argument: ``%d``, position: -1
        mutually_exclusive: in_file2, in_float
in_float: (a float)
        Input float
        argument: ``%f``, position: -1
        mutually_exclusive: in_file2, in_int
in_file2: (an existing file name)
        Input file 2
        argument: ``%s``, position: -1
        mutually_exclusive: in_float, in_int
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
subjects_dir: (an existing directory name)
        subjects directory

```

Outputs:

```

out_file: (a file name)
        Output file after calculation

```

62.5.19 MRIsCombine

Link to code

Wraps the executable command `mriscombine`.

Uses `FreeSurfer's mriscombine` to combine two surface files into one.

For complete details, see the [mriscombine Documentation](#).

If given an `out_file` that does not begin with `'lh.'` or `'rh.'`, `mriscombine` 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
'mriscombine --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)

```

(continues on next page)

(continued from previous page)

```

    Two surfaces to be combined.
    argument: ``--combinesurfs %s``, position: 1
out_file: (a file name)
    Output filename. Combined surfaces from in_files.
    argument: ``%s``, position: -1

[Optional]
subjects_dir: (an existing directory name)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

out_file: (an existing file name)
    Output filename. Combined surfaces from in_files.

```

62.5.20 MRIsConvert[Link to code](#)Wraps the executable command `mrisc`.Uses Freesurfer's `mrisc` 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
    argument: ``%s``, position: -2
out_file: (a file name)
    output filename or True to generate one
    argument: ``%s``, position: -1
    mutually_exclusive: out_datatype
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

[Optional]
rescale: (a boolean)
    rescale vertex xyz so total area is same as group average

```

(continues on next page)

(continued from previous page)

```

        argument: ``-r``
normal: (a boolean)
        output is an ascii file where vertex data
        argument: ``-n``
to_tkr: (a boolean)
        convert coordinates from scanner coords to native FS (tkr) coords
        argument: ``--to-tkr``
talairachxfm_subjid: (a string)
        apply talairach xfm of subject to vertex xyz
        argument: ``-t %s``
dataarray_num: (an integer (int or long))
        if input is gifti, 'num' specifies which data array to use
        argument: ``--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
label_file: (an existing file name)
        infile is .label file, label is name of this label
        argument: ``--label %s``
functional_file: (an existing file name)
        input is functional time-series or other multi-frame data (must
        specify surface)
        argument: ``-f %s``
labelstats_outfile: (a file name)
        outfile is name of gifti file to which label stats will be written
        argument: ``--labelstats %s``
patch: (a boolean)
        input is a patch, not a full surface
        argument: ``-p``
vertex: (a boolean)
        Writes out neighbors of a vertex in each row
        argument: ``-v``
subjects_dir: (an existing directory name)
        subjects directory
annot_file: (an existing file name)
        input is annotation or gifti label data
        argument: ``--annot %s``
xyz_ascii: (a boolean)
        Print only surface xyz to ascii file
        argument: ``-a``
to_scanner: (a boolean)
        convert coordinates from native FS (tkr) coords to scanner coords
        argument: ``--to-scanner``
scale: (a float)
        scale vertex xyz by scale
        argument: ``-s %.3f``
scalarcuv_file: (an existing file name)
        input is scalar curv overlay file (must still specify surface)
        argument: ``-c %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
origname: (a string)
        read orig positions
        argument: ``-o %s``
parcstats_file: (an existing file name)

```

(continues on next page)

(continued from previous page)

```
infile is name of text file containing label/val pairs
argument: ``--parcstats %s``
```

Outputs:

```
converted: (an existing file name)
          converted output surface
```

62.5.21 MRIsExpand[Link to code](#)Wraps the executable command `mriss_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]
in_file: (an existing file name)
         Surface to expand
         argument: ``%s``, position: -3
distance: (a float)
         Distance in mm or fraction of cortical thickness
         argument: ``%g``, position: -2

[Optional]
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`
               argument: ``-thickness_name %s``
write_iterations: (an integer (int or long))
                 Write snapshots of expansion every N iterations
                 argument: ``-W %d``
subjects_dir: (an existing directory name)
              subjects directory
nsurfaces: (an integer (int or long))
           Number of surfaces to write during expansion
           argument: ``-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`
          argument: ``%s``, position: -1
sphere: (a unicode string, nipy default value: sphere)
        WARNING: Do not change this trait
```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
thickness: (a boolean)
           Expand by fraction of cortical thickness, not mm
           argument: ``-thickness``
dt: (a float)
    dt (implicit: 0.25)
    argument: ``-T %g``
spring: (a float)
        Spring term (implicit: 0.05)
        argument: ``-S %g``
smooth_averages: (an integer (int or long))
                 Smooth surface with N iterations after expansion
                 argument: ``-A %d``
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`
      argument: ``-pial %s``

```

Outputs:

```

out_file: (a file name)
          Output surface file

```

62.5.22 MRIsInflate[Link to code](#)Wraps the executable command `mris_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
         argument: ``%s``, position: -2

[Optional]
out_file: (a file name)
          Output file for MRIsInflate
          argument: ``%s``, position: -1

```

(continues on next page)

(continued from previous page)

```

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
out_sulc: (a file name)
    Output sulc file
    mutually_exclusive: no_save_sulc
no_save_sulc: (a boolean)
    Do not save sulc file as output
    argument: ``-no-save-sulc``
    mutually_exclusive: out_sulc
environ: (a dictionary with keys which are a 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
subjects_dir: (an existing directory name)
    subjects directory

```

Outputs:

```

out_sulc: (a file name)
    Output sulc file
out_file: (a file name)
    Output file for MRIsInflate

```

62.5.23 MakeAverageSubject[Link to code](#)Wraps the executable command `make_average_subject`.

Make an average freesurfer subject

Examples

```

>>> from nipype.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
    argument: ``--subjects %s``

[Optional]
out_name: (a file name, nipype default value: average)
    name for the average subject
    argument: ``--out %s``
subjects_dir: (an existing directory name)
    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a

```

(continues on next page)

(continued from previous page)

```
value of class 'str', nipyne default value: {}
Environment variables
```

Outputs:

```
average_subject_name: (a unicode string)
Output registration file
```

62.5.24 MakeSurfaces[Link to code](#)

Wraps the executable 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 nipyne.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]
in_orig: (an existing file name)
    Implicit input file <hemisphere>.orig
    argument: ``-orig %s``
subject_id: (a string, nipyne default value: subject_id)
    Subject being processed
    argument: ``%s``, position: -2
in_wm: (an existing file name)
    Implicit input file wm.mgz
in_filled: (an existing file name)
    Implicit input file filled.mgz
hemisphere: ('lh' or 'rh')
    Hemisphere being processed
    argument: ``%s``, position: -1

[Optional]
in_label: (an existing file name)
    Implicit input label/<hemisphere>.aparc.annot
    mutually_exclusive: noaparc
in_T1: (an existing file name)
    Input brain or T1 file
    argument: ``-T1 %s``
environ: (a dictionary with keys which are a bytes or None or a value
```

(continues on next page)

(continued from previous page)

```

        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipyype default value: {}))
    Environment variables
fix_mtl: (a boolean)
    Undocumented flag
    argument: ``-fix_mtl``
mgz: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    argument: ``-mgz``
in_aseg: (an existing file name)
    Input segmentation file
    argument: ``-aseg %s``
in_white: (an existing file name)
    Implicit input that is sometimes used
orig_pial: (an existing file name)
    Specify a pial surface to start with
    argument: ``-orig_pial %s``
    requires: in_label
orig_white: (an existing file name)
    Specify a white surface to start with
    argument: ``-orig_white %s``
subjects_dir: (an existing directory name)
    subjects directory
longitudinal: (a boolean)
    No documentation (used for longitudinal processing)
    argument: ``-long``
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the input
    files to the node directory.
white: (a string)
    White surface name
    argument: ``-white %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
noaparc: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    argument: ``-noaparc``
    mutually_exclusive: in_label
maximum: (a float)
    No documentation (used for longitudinal processing)
    argument: ``-max %.1f``
white_only: (a boolean)
    Undocumented flage
    argument: ``-whiteonly``
no_white: (a boolean)
    Undocumented flag
    argument: ``-nowhite``

```

Outputs:

```

out_cortex: (a file name)
    Output cortex file for MakeSurfaces
out_thickness: (a file name)
    Output thickness file for MakeSurfaces

```

(continues on next page)

(continued from previous page)

```

out_curv: (a file name)
    Output curv file for MakeSurfaces
out_pial: (a file name)
    Output pial surface for MakeSurfaces
out_white: (a file name)
    Output white matter hemisphere surface
out_area: (a file name)
    Output area file for MakeSurfaces

```

62.5.25 ParcellationStats

[Link to code](#)

Wraps the executable command `mris_anatomical_stats`.

This program computes a number of anatomical properties.

Examples

```

>>> from nipyre.interfaces.freesurfer import ParcellationStats
>>> import os
>>> parcellationstats = ParcellationStats()
>>> parcellationstats.inputs.subject_id = '10335'
>>> parcellationstats.inputs.hemisphere = 'lh'
>>> parcellationstats.inputs.wm = '../mri/wm.mgz'
>>> parcellationstats.inputs.transform = '../mri/transforms/talairach.xfm'
>>> parcellationstats.inputs.brainmask = '../mri/brainmask.mgz'
>>> parcellationstats.inputs.aseg = '../mri/aseg.presurf.mgz'
>>> parcellationstats.inputs.ribbon = '../mri/ribbon.mgz'
>>> parcellationstats.inputs.lh_pial = 'lh.pial'
>>> parcellationstats.inputs.rh_pial = 'rh.pial'
>>> parcellationstats.inputs.lh_white = 'lh.white'
>>> parcellationstats.inputs.rh_white = 'rh.white'
>>> parcellationstats.inputs.thickness = 'lh.thickness'
>>> parcellationstats.inputs.surface = 'white'
>>> parcellationstats.inputs.out_table = 'lh.test.stats'
>>> parcellationstats.inputs.out_color = 'test.ctab'
>>> parcellationstats.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
rh_pial: (an existing file name)
    Input file must be <subject_id>/surf/rh.pial
ribbon: (an existing file name)
    Input file must be <subject_id>/mri/ribbon.mgz
transform: (an existing file name)
    Input file must be <subject_id>/mri/transforms/talairach.xfm
thickness: (an existing file name)
    Input file must be <subject_id>/surf/?h.thickness
rh_white: (an existing file name)
    Input file must be <subject_id>/surf/rh.white
subject_id: (a string, nipyre default value: subject_id)
    Subject being processed

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``, position: -3
lh_pial: (an existing file name)
    Input file must be <subject_id>/surf/lh.pial
hemisphere: ('lh' or 'rh')
    Hemisphere being processed
        argument: ``%s``, position: -2
lh_white: (an existing file name)
    Input file must be <subject_id>/surf/lh.white
wm: (an existing file name)
    Input file must be <subject_id>/mri/wm.mgz
brainmask: (an existing file name)
    Input file must be <subject_id>/mri/brainmask.mgz

[Optional]
in_cortex: (a file name)
    Input cortex label
    argument: ``-cortex %s``
surface: (a string)
    Input surface (e.g. 'white')
    argument: ``%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', nipyype default value: {})
    Environment variables
mgz: (a boolean)
    Look for mgz files
    argument: ``-mgz``
in_label: (a file name)
    limit calculations to specified label
    argument: ``-l %s``
    mutually_exclusive: in_annotation, out_color
out_table: (a file name)
    Table output to tablefile
    argument: ``-f %s``
    requires: tabular_output
out_color: (a file name)
    Output annotation files's colortable to text file
    argument: ``-c %s``
    mutually_exclusive: in_label
subjects_dir: (an existing directory name)
    subjects directory
th3: (a boolean)
    turns on new vertex-wise volume calc for mris_anat_stats
    argument: ``-th3``
    requires: cortex_label
copy_inputs: (a boolean)
    If running as a node, set this to True. This will copy the input
    files to the node directory.
in_annotation: (a file name)
    compute properties for each label in the annotation file separately
    argument: ``-a %s``
    mutually_exclusive: in_label
tabular_output: (a boolean)
    Tabular output
    argument: ``-b``
args: (a unicode string)
    Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
cortex_label: (an existing file name)
        implicit input file {hemi}.cortex.label

```

Outputs:

```

out_table: (a file name)
        Table output to tablefile
out_color: (a file name)
        Output annotation files's colortable to text file

```

62.5.26 RelabelHypointensities[Link to code](#)Wraps the executable 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
        argument: ``%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]
out_file: (a file name)
        Output aseg file
        argument: ``%s``, position: -1
subjects_dir: (an existing directory name)
        subjects directory
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
surf_directory: (a directory name, nipy default value: .)
        Directory containing lh.white and rh.white
        argument: ``%s``, position: -2

```

Outputs:

```
out_file: (a file name)
          Output aseg file
          argument: ``%s``
```

62.5.27 RemoveIntersection

[Link to code](#)

Wraps the executable command `mris_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
          argument: ``%s``, position: -2

[Optional]
out_file: (a file name)
          Output file for RemoveIntersection
          argument: ``%s``, position: -1
subjects_dir: (an existing directory name)
              subjects directory
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
```

Outputs:

```
out_file: (a file name)
          Output file for RemoveIntersection
```

62.5.28 RemoveNeck

[Link to code](#)

Wraps the executable 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'
```

(continues on next page)

(continued from previous page)

```
>>> 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]
transform: (an existing file name)
    Input transform file for RemoveNeck
    argument: ``%s``, position: -3
in_file: (an existing file name)
    Input file for RemoveNeck
    argument: ``%s``, position: -4
template: (an existing file name)
    Input template file for RemoveNeck
    argument: ``%s``, position: -2

[Optional]
out_file: (a file name)
    Output file for RemoveNeck
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
radius: (an integer (int or long))
    Radius
    argument: ``-radius %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
subjects_dir: (an existing directory name)
    subjects directory
```

Outputs:

```
out_file: (a file name)
    Output file with neck removed
```

62.5.29 SampleToSurface[Link to code](#)

Wraps the executable 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
    argument: ``--hemi %s``
projection_stem: (a string)
    stem for precomputed linear estimates and volume fractions
    mutually_exclusive: sampling_method
sampling_method: ('point' or 'max' or 'average')
    how to sample -- at a point or at the max or average over a range
    argument: ``%s``
    mutually_exclusive: projection_stem
    requires: sampling_range, sampling_units
reg_file: (an existing file name)
    source-to-reference registration file
    argument: ``--reg %s``
    mutually_exclusive: reg_file, reg_header, mni152reg
reg_header: (a boolean)
    register based on header geometry
    argument: ``--regheader %s``
    mutually_exclusive: reg_file, reg_header, mni152reg
    requires: subject_id
source_file: (an existing file name)
    volume to sample values from
    argument: ``--mov %s``
mni152reg: (a boolean)
    source volume is in MNI152 space
    argument: ``--mni152reg``
    mutually_exclusive: reg_file, reg_header, mni152reg

[Optional]
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'
reference_file: (an existing file name)
    reference volume (default is orig.mgz)
    argument: ``--ref %s``
target_subject: (a string)
    sample to surface of different subject than source
    argument: ``--trgs subject %s``
ico_order: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

        icosahedron order when target_subject is 'ico'
        argument: ``--icoorder %d``
        requires: target_subject
fixTkReg: (a boolean)
    make reg matrix round-compatible
    argument: ``--fixtkreg``
noReshape: (a boolean)
    do not reshape surface vector (default)
    argument: ``--noreshape``
    mutually_exclusive: reshape
overrideRegSubj: (a boolean)
    override the subject in the reg file header
    argument: ``--srcsubject %s``
    requires: subject_id
surface: (a string)
    target surface (default is white)
    argument: ``--surf %s``
environ: (a dictionary with keys which are a 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
cortexMask: (a boolean)
    mask the target surface with hemi.cortex.label
    argument: ``--cortex``
    mutually_exclusive: mask_label
reshapeSlices: (an integer (int or long))
    number of 'slices' for reshaping
    argument: ``--rf %d``
voxFile: (a boolean or a file name)
    text file with the number of voxels intersecting the surface
    argument: ``--nvox %s``
hitsType: ('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
    argument: ``--srchit_type``
surfReg: (a boolean or a unicode string)
    use surface registration to target subject
    argument: ``--surfreg %s``
    requires: target_subject
subjectsDir: (an existing directory name)
    subjects directory
smoothSurf: (a float)
    smooth output surface (mm fwhm)
    argument: ``--surf-fwhm %.3f``
float2intMethod: ('round' or 'tkregister')
    method to convert reg matrix values (default is round)
    argument: ``--float2int %s``
outFile: (a file name)
    surface file to write
    argument: ``--o %s``
applyTrans: (a tuple of the form: (a float, a float, a float))
    translation (in mm) to apply to reg matrix
    argument: ``--trans %.3f %.3f %.3f``
hitsFile: (a boolean or an existing file name)
    save image with number of hits at each voxel

```

(continues on next page)

(continued from previous page)

```

    argument: ``--srchit %s``
frame: (an integer (int or long))
    save only one frame (0-based)
    argument: ``--frame %d``
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
    argument: ``--out_type %s``
apply_rot: (a tuple of the form: (a float, a float, a float))
    rotation angles (in degrees) to apply to reg matrix
    argument: ``--rot %.3f %.3f %.3f``
subject_id: (a string)
    subject id
interp_method: ('nearest' or 'trilinear')
    interpolation method
    argument: ``--interp %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
smooth_vol: (a float)
    smooth input volume (mm fwhm)
    argument: ``--fwhm %.3f``
scale_input: (a float)
    multiple all intensities by scale factor
    argument: ``--scale %.3f``
reshape: (a boolean)
    reshape surface vector to fit in non-mgh format
    argument: ``--reshape``
    mutually_exclusive: no_reshape
mask_label: (an existing file name)
    label file to mask output with
    argument: ``--mask %s``
    mutually_exclusive: cortex_mask

```

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

```

62.5.30 SmoothTessellation[Link to code](#)Wraps the executable 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.
        argument: ``%s``, position: -2

[Optional]
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
use_gaussian_curvature_smoothing: (a boolean)
      Use Gaussian curvature smoothing
      argument: ``-g``
out_file: (a file name)
          output filename or True to generate one
          argument: ``%s``, position: -1
smoothing_iterations: (an integer (int or long))
                      Number of smoothing iterations (default=10)
                      argument: ``-n %d``
disable_estimates: (a boolean)
                   Disables the writing of curvature and area estimates
                   argument: ``-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
out_area_file: (a file name)
               Write area to ?h.areaname (default "area")
               argument: ``-b %s``
use_momentum: (a boolean)
               Uses momentum
               argument: ``-m``
seed: (an integer (int or long))
      Seed for setting random number generator
      argument: ``-seed %d``
snapshot_writing_iterations: (an integer (int or long))
                             Write snapshot every "n" iterations
                             argument: ``-w %d``
gaussian_curvature_smoothing_steps: (an integer (int or long))
                                     Use Gaussian curvature smoothing
                                     argument: ``%d``
curvature_averaging_iterations: (an integer (int or long))
                                Number of curvature averaging iterations (default=10)
                                argument: ``-a %d``
normalize_area: (a boolean)
                Normalizes the area after smoothing
                argument: ``-area``
out_curvature_file: (a file name)
                    Write curvature to ?h.curvname (default "curv")
                    argument: ``-c %s``
```

(continues on next page)

(continued from previous page)

```

gaussian_curvature_norm_steps: (an integer (int or long))
    Use Gaussian curvature smoothing
    argument: ``%d``
subjects_dir: (an existing directory name)
    subjects directory

```

Outputs:

```

surface: (an existing file name)
    Smoothed surface file

```

62.5.31 Sphere[Link to code](#)Wraps the executable command `mris_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
    argument: ``%s``, position: -2

[Optional]
seed: (an integer (int or long))
    Seed for setting random number generator
    argument: ``-seed %d``
subjects_dir: (an existing directory name)
    subjects directory
out_file: (a file name)
    Output file for Sphere
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
in_smoothwm: (an existing file name)
    Input surface required when -q flag is not selected
num_threads: (an integer (int or long))
    allows for specifying more threads
magic: (a boolean)
    No documentation. Direct questions to analysis-
    bugs@nmr.mgh.harvard.edu
    argument: ``-q``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```


Outputs:

```
out_file: (a file name)
          Output file for Sphere
```

62.5.32 Surface2VolTransform[Link to code](#)Wraps the executable 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.copel.mgz'
>>> xfm2vol.inputs.reg_file = 'register.mat'
>>> xfm2vol.inputs.hemi = 'lh'
>>> xfm2vol.inputs.template_file = 'copel.nii.gz'
>>> xfm2vol.inputs.subjects_dir = '.'
>>> xfm2vol.cmdline
'mri_surf2vol --hemi lh --volreg register.mat --surfval lh.copel.mgz --sd . --
↳template copel.nii.gz --outvol lh.copel_asVol.nii --vtxvol lh.copel_asVol_
↳vertex.nii'
>>> res = xfm2vol.run()
```

Inputs:

```
[Mandatory]
hemi: (a unicode string)
      hemisphere of data
      argument: ``--hemi %s``
reg_file: (an existing file name)
          tkRAS-to-tkRAS matrix (tkregister2 format)
          argument: ``--volreg %s``
          mutually_exclusive: subject_id
source_file: (an existing file name)
             This is the source of the surface values
             argument: ``--surfval %s``
             mutually_exclusive: mkmask

[Optional]
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.
               argument: ``--vtxvol %s``
transformed_file: (a file name)
                 Output volume
                 argument: ``--outvol %s``
template_file: (an existing file name)
               Output template volume
               argument: ``--template %s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
```

(continues on next page)

(continued from previous page)

```

surf_name: (a unicode string)
    surfname (default is white)
    argument: ``--surf %s``
subject_id: (a unicode string)
    subject id
    argument: ``--identity %s``
    mutually_exclusive: reg_file
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
mkmask: (a boolean)
    make a mask instead of loading surface values
    argument: ``--mkmask``
    mutually_exclusive: source_file
projfrac: (a float)
    thickness fraction
    argument: ``--projfrac %s``
subjects_dir: (a unicode string)
    freesurfer subjects directory defaults to $SUBJECTS_DIR
    argument: ``--sd %s``

```

Outputs:

```

vertexvol_file: (a file name)
    vertex map volume path id. Optional
transformed_file: (an existing file name)
    Path to output file if used normally

```

62.5.33 SurfaceSmooth

[Link to code](#)

Wraps the executable command `mri_surf2surf`.

Smooth a surface image with `mri_surf2surf`.

The surface is smoothed by an interactive 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 nipy.interfaces.freesurfer as fs
>>> smoother = fs.SurfaceSmooth()
>>> smoother.inputs.in_file = "lh.copel.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.copel.mgz --tval ...lh.
↳ copel_smooth5.mgz --s subj_1'
>>> smoother.run()

```

Inputs:

```

[Mandatory]
subject_id: (a string)

```

(continues on next page)

(continued from previous page)

```

        subject id of surface file
        argument: ``--s %s``
hemi: ('lh' or 'rh')
        hemisphere to operate on
        argument: ``--hemi %s``
in_file: (a file name)
        source surface file
        argument: ``--sval %s``

[Optional]
out_file: (a file name)
        surface file to write
        argument: ``--tval %s``
subjects_dir: (an existing directory name)
        subjects directory
cortex: (a boolean, nipype default value: True)
        only smooth within $hemi.cortex.label
        argument: ``--cortex``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
smooth_iters: (an integer (int or long))
        iterations of the smoothing process
        argument: ``--smooth %d``
        mutually_exclusive: fwhm
reshape: (a boolean)
        reshape surface vector to fit in non-mgh format
        argument: ``--reshape``
environ: (a dictionary with keys which are a 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
fwhm: (a float)
        effective FWHM of the smoothing process
        argument: ``--fwhm %.4f``
        mutually_exclusive: smooth_iters

```

Outputs:

```

out_file: (an existing file name)
        smoothed surface file

```

62.5.34 SurfaceSnapshots[Link to code](#)

Wraps the executable 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.

Note that this interface will not run if you do not have graphics enabled on your system.

Examples

```
>>> import nipyre.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]
subject_id: (a string)
    subject to visualize
    argument: ``%s``, position: 1
hemi: ('lh' or 'rh')
    hemisphere to visualize
    argument: ``%s``, position: 2
surface: (a string)
    surface to visualize
    argument: ``%s``, position: 3

[Optional]
six_images: (a boolean)
    also take anterior and posterior snapshots
invert_overlay: (a boolean)
    invert the overlay display
    argument: ``-invphaseflag 1``
show_curv: (a boolean)
    show curvature
    argument: ``-curv``
    mutually_exclusive: show_gray_curv
mni152_reg: (a boolean)
    use to display a volume in MNI152 space on the average subject
    argument: ``-mni152reg``
    mutually_exclusive: overlay_reg, identity_reg, mni152_reg
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)
    argument: ``%s``
overlay: (an existing file name)
    load an overlay volume/surface
    argument: ``-overlay %s``
    requires: overlay_range
environ: (a dictionary with keys which are a 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
label_file: (an existing file name)
    path to label file to display
    argument: ``-label %s``
    mutually_exclusive: label_name
show_gray_curv: (a boolean)
    show curvature in gray
    argument: ``-gray``
    mutually_exclusive: show_curv
label_under: (a boolean)
    draw label/annotation under overlay
```

(continues on next page)

(continued from previous page)

```

        argument: ``-labels-under``
label_outline: (a boolean)
    draw label/annotation as outline
    argument: ``-label-outline``
overlay_reg: (a file name)
    registration matrix file to register overlay to surface
    argument: ``-overlay-reg %s``
    mutually_exclusive: overlay_reg, identity_reg, mni152_reg
show_color_text: (a boolean)
    display text in the color scale bar
    argument: ``-colscaletext 1``
reverse_overlay: (a boolean)
    reverse the overlay display
    argument: ``-revphaseflag 1``
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
patch_file: (an existing file name)
    load a patch
    argument: ``-patch %s``
subjects_dir: (an existing directory name)
    subjects directory
truncate_overlay: (a boolean)
    truncate the overlay display
    argument: ``-truncphaseflag 1``
demean_overlay: (a boolean)
    remove mean from overlay
    argument: ``-zm``
annot_file: (an existing file name)
    path to annotation file to display
    argument: ``-annotation %s``
    mutually_exclusive: annot_name
orig_suffix: (a string)
    set the orig surface suffix string
    argument: ``-orig %s``
identity_reg: (a boolean)
    use the identity matrix to register the overlay to the surface
    argument: ``-overlay-reg-identity``
    mutually_exclusive: overlay_reg, identity_reg, mni152_reg
tcl_script: (an existing file name)
    override default screenshot script
    argument: ``%s``
label_name: (a string)
    name of label to display (must be in $subject/label directory
    argument: ``-label %s``
    mutually_exclusive: label_file
colortable: (an existing file name)
    load colortable file
    argument: ``-colortable %s``
sphere_suffix: (a string)
    set the sphere.reg suffix string
    argument: ``-sphere %s``
annot_name: (a string)
    name of annotation to display (must be in $subject/label directory
    argument: ``-annotation %s``
    mutually_exclusive: annot_file
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

        Additional parameters to the command
        argument: ``%s``
show_color_scale: (a boolean)
    display the color scale bar
    argument: ``-colscalebarflag 1``
screenshot_stem: (a string)
    stem to use for screenshot file names
overlay_range_offset: (a float)
    overlay range will be symettric around offset value
    argument: ``-foffset %.3f``

```

Outputs:

```

snapshots: (a list of items which are an existing file name)
    tiff images of the surface from different perspectives

```

62.5.35 SurfaceTransform[Link to code](#)

Wraps the executable 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.cope1.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
    argument: ``--hemi %s``
source_subject: (a string)
    subject id for source surface
    argument: ``--srcsubject %s``
source_annot_file: (an existing file name)
    surface annotation file
    argument: ``--sval-annot %s``
    mutually_exclusive: source_file
target_subject: (a string)
    subject id of target surface
    argument: ``--trgssubject %s``
source_file: (an existing file name)
    surface file with source values
    argument: ``--sval %s``
    mutually_exclusive: source_annot_file

[Optional]

```

(continues on next page)

(continued from previous page)

```

out_file: (a file name)
    surface file to write
    argument: ``--tval %s``
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'
    argument: ``--trgicoorder %d``
reshape: (a boolean)
    reshape output surface to conform with Nifti
    argument: ``--reshape``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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 'nifti1' or 'nii' or
    'niigz')
    source file format
    argument: ``--sfmt %s``
    requires: source_file
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
reshape_factor: (an integer (int or long))
    number of slices in reshaped image
    argument: ``--reshape-factor``
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
    argument: ``--tfmt %s``
subjects_dir: (an existing directory name)
    subjects directory

```

Outputs:

```

out_file: (an existing file name)
    transformed surface file

```

62.5.36 TalairachAVI[Link to code](#)

Wraps the executable 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
        argument: ``--i %s``
out_file: (a file name)
        output xfm file
        argument: ``--xfm %s``

[Optional]
subjects_dir: (an existing directory name)
              subjects directory
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
atlas: (a string)
       alternate target atlas (in freesurfer/average dir)
       argument: ``--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
```

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
```

62.5.37 TalairachQC

[Link to code](#)

Wraps the executable 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
          argument: ``%s``, position: 0

[Optional]
subjects_dir: (an existing directory name)
```

(continues on next page)

(continued from previous page)

```

    subjects directory
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

log_file: (an existing file name, nipy default value:
    output.nipy)
    The output log

```

62.5.38 Tkregister2

[Link to code](#)

Wraps the executable 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]
reg_file: (a file name, nipy default value: register.dat)
    freesurfer-style registration file
    argument: ``--reg %s``
moving_image: (an existing file name)
    moving volume
    argument: ``--mov %s``

[Optional]

```

(continues on next page)

(continued from previous page)

```

movscale: (a float)
    adjust registration matrix to scale mov
    argument: ``--movscale %f``
target_image: (an existing file name)
    target volume
    argument: ``--targ %s``
    mutually_exclusive: fstarg
lta_out: (a bool or None or a file name)
    output registration file (LTA format)
    argument: ``--ltaout %s``
subjects_dir: (an existing directory name)
    subjects directory
fsl_in_matrix: (an existing file name)
    fsl-style registration input matrix
    argument: ``--fsl %s``
invert_lta_out: (a boolean)
    Invert input LTA before applying
    argument: ``--ltaout-inv``
    requires: lta_in
xfm: (an existing file name)
    use a matrix in MNI coordinates as initial registration
    argument: ``--xfm %s``
environ: (a dictionary with keys which are a 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
subject_id: (a string)
    freesurfer subject ID
    argument: ``--s %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
fstarg: (a boolean)
    use subject's T1 as reference
    argument: ``--fstarg``
    mutually_exclusive: target_image
invert_lta_in: (a boolean)
    Invert input LTA before applying
    requires: lta_in
lta_in: (an existing file name)
    use a matrix in MNI coordinates as initial registration
    argument: ``--lta %s``
fsl_out: (a bool or None or a file name)
    compute an FSL-compatible resgitration matrix
    argument: ``--fslregout %s``
reg_header: (a boolean)
    compute regstration from headers
    argument: ``--regheader``
fstal: (a boolean)
    set mov to be tal and reg to be tal xfm
    argument: ``--fstal``
    mutually_exclusive: target_image, moving_image, reg_file
noedit: (a boolean, nipyte default value: True)
    do not open edit window (exit)
    argument: ``--noedit``

```

Outputs:

```
lta_file: (a file name)
    LTA-style registration file
reg_file: (an existing file name)
    freesurfer-style registration file
fsl_file: (a file name)
    FSL-style registration file
```

62.5.39 VolumeMask

[Link to code](#)

Wraps the executable command `mris_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].[whitel|pial]` and labels voxels based on the signed-distance function from the surface.

Examples

```
>>> from nipy.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]
subject_id: (a string, nipy default value: subject_id)
    Subject being processed
    argument: ``%s``, position: -1
right_whitelabel: (an integer (int or long))
    Right white matter label
    argument: ``--label_right_white %d``
lh_pial: (an existing file name)
    Implicit input left pial surface
rh_pial: (an existing file name)
    Implicit input right pial surface
right_ribbonlabel: (an integer (int or long))
    Right cortical ribbon label
    argument: ``--label_right_ribbon %d``
left_whitelabel: (an integer (int or long))
    Left white matter label
    argument: ``--label_left_white %d``
lh_white: (an existing file name)
    Implicit input left white matter surface
left_ribbonlabel: (an integer (int or long))
```

(continues on next page)

(continued from previous page)

```

    Left cortical ribbon label
    argument: ``--label_left_ribbon %d``
rh_white: (an existing file name)
    Implicit input right white matter surface

[Optional]
aseg: (an existing file name)
    Implicit aseg.mgz segmentation. Specify a different aseg by using
    the 'in_aseg' input.
    mutually_exclusive: in_aseg
save_ribbon: (a boolean)
    option to save just the ribbon for the hemispheres in the format
    ?h.ribbon.mgz
    argument: ``--save_ribbon``
subjects_dir: (an existing directory name)
    subjects directory
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', nipy default value: {})
    Environment variables
in_aseg: (an existing file name)
    Input aseg file for VolumeMask
    argument: ``--aseg_name %s``
    mutually_exclusive: aseg
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

```

Outputs:

```

out_ribbon: (a file name)
    Output cortical ribbon mask
lh_ribbon: (a file name)
    Output left cortical ribbon mask
rh_ribbon: (a file name)
    Output right cortical ribbon mask

```

62.5.40 copy2subjdir()[Link to code](#)

Method to copy an input to the subjects directory

62.5.41 createoutputdirs()[Link to code](#)

create all output directories. If not created, some freesurfer interfaces fail

63.1 interfaces.fsl.aroma

63.1.1 ICA_AROMA

[Link to code](#)

Wraps the executable 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()
>>> 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]
motion_parameters: (an existing file name)
                   motion parameters file
                   argument: ``-mc %s``
```

(continues on next page)

(continued from previous page)

```

    mutually_exclusive: feat_dir
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.
    argument: ``-feat %s``
    mutually_exclusive: in_file, mat_file, fnirt_warp_file,
    motion_parameters
in_file: (an existing file name)
    volume to be denoised
    argument: ``-i %s``
    mutually_exclusive: feat_dir
denoise_type: ('nonaggr' or 'aggr' or 'both' or 'no', nipy default
    value: nonaggr)
    Type of denoising strategy:
    -no: 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)
    argument: ``-den %s``
out_dir: (a directory name, nipy default value: out)
    output directory
    argument: ``-o %s``

[Optional]
mask: (an existing file name)
    path/name volume mask
    argument: ``-m %s``
    mutually_exclusive: feat_dir
melodic_dir: (an existing directory name)
    path to MELODIC directory if MELODIC has already been run
    argument: ``-meldir %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
dim: (an integer (int or long))
    Dimensionality reduction when running MELODIC (default is automatic
    estimation)
    argument: ``-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
TR: (a float)
    TR in seconds. If this is not specified the TR will be extracted
    from the header of the fMRI nifti file.
    argument: ``-tr %.3f``
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)
    argument: ``-affmat %s``
    mutually_exclusive: feat_dir
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)
    argument: ``-warp %s``
    mutually_exclusive: feat_dir

```

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)

```

63.2 interfaces.fsl.dti

63.2.1 BEDPOSTX5

[Link to code](#)

Wraps the executable 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 `nipyne.workflows.fsl.dmri.create_bedpostx_pipeline()` instead.

Example

```

>>> from nipyne.interfaces import fsl
>>> bedp = fsl.BEDPOSTX5(bvecs='bvecs', bvals='bvals', dwi='diffusion.nii',
...                      mask='mask.nii', n_fibres=1)
>>> bedp.cmdline
'bedpostx bedpostx -b 0 --burnin_noard=0 --forcedir -n 1 -j 5000 -s 1 --
↪updateproposalevery=40'

```

Inputs:

```

[Mandatory]
mask: (an existing file name)
    bet binary mask file
dwi: (an existing file name)
    diffusion weighted image data file
bvecs: (an existing file name)
    b vectors file
n_fibres: (a long integer >= 1, nipyne default value: 2)
    Maximum number of fibres to fit in each voxel
    argument: ``-n %d``
out_dir: (a directory name, nipyne default value: bedpostx)
    output directory
    argument: ``%s``, position: 1
bvals: (an existing file name)
    b values file

[Optional]
n_jumps: (an integer (int or long), nipyne default value: 5000)
    Num of jumps to be made by MCMC

```

(continues on next page)

(continued from previous page)

```

        argument: ``-j %d``
f0_ard: (a boolean)
    Noise floor model: add to the model an unattenuated signal
    compartment f0
    argument: ``--f0 --ardf0``
    mutually_exclusive: f0_noard, f0_ard, all_ard
logdir: (a directory name)
    argument: ``--logdir=%s``
all_ard: (a boolean)
    Turn ARD on on all fibres
    argument: ``--allard``
    mutually_exclusive: no_ard, all_ard
use_gpu: (a boolean)
    Use the GPU version of bedpostx
fudge: (an integer (int or long))
    ARD fudge factor
    argument: ``-w %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', nipyne default value: {})
    Environment variables
f0_noard: (a boolean)
    Noise floor model: add to the model an unattenuated signal
    compartment f0
    argument: ``--f0``
    mutually_exclusive: f0_noard, f0_ard
seed: (an integer (int or long))
    seed for pseudo random number generator
    argument: ``--seed=%d``
no_spat: (a boolean)
    Initialise with tensor, not spatially
    argument: ``--nospat``
    mutually_exclusive: no_spat, non_linear, clinear
no_ard: (a boolean)
    Turn ARD off on all fibres
    argument: ``--noard``
    mutually_exclusive: no_ard, all_ard
force_dir: (a boolean, nipyne default value: True)
    use the actual directory name given (do not add + to make a new
    directory)
    argument: ``--forcedir``
sample_every: (a long integer >= 0, nipyne default value: 1)
    Num of jumps for each sample (MCMC)
    argument: ``-s %d``
burn_in_noard: (a long integer >= 0, nipyne default value: 0)
    num of burnin jumps before the ard is imposed
    argument: ``--burnin_noard=%d``
grad_dev: (an existing file name)
    grad_dev file, if gradnonlin, -g is True
update_proposal_every: (a long integer >= 1, nipyne default value:
    40)
    Num of jumps for each update to the proposal density std (MCMC)
    argument: ``--updateproposalevery=%d``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
clinear: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        Initialise with constrained nonlinear fitting
        argument: ``--cnonlinear``
        mutually_exclusive: no_spat, non_linear, cnonlinear
    rician: (a boolean)
        use Rician noise modeling
        argument: ``--rician``
    model: (1 or 2 or 3)
        use monoexponential (1, default, required for single-shell) or
        multiexponential (2, multi-shell) model
        argument: ``-model %d``
    args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
    gradnonlin: (a boolean)
        consider gradient nonlinearities, default off
        argument: ``-g``
    non_linear: (a boolean)
        Initialise with nonlinear fitting
        argument: ``--nonlinear``
        mutually_exclusive: no_spat, non_linear, cnonlinear
    burn_in: (a long integer >= 0, nipy default value: 0)
        Total num of jumps at start of MCMC to be discarded
        argument: ``-b %d``

```

Outputs:

```

merged_fsamples: (a list of items which are an existing file name)
    Samples from the distribution on anisotropic volume fraction
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
merged_phsamples: (a list of items which are an existing file name)
    Samples from the distribution on phi
dyads_dispersions: (a list of items which are an existing file name)
    Dispersion
merged_thsamples: (a list of items which are an existing file name)
    Samples from the distribution on theta
dyads: (a list of items which are an existing file name)
    Mean of PDD distribution in vector form.
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
mean_fsamples: (a list of items which are an existing file name)
    Mean of distribution on f anisotropy

```

References:

None

63.2.2 DTIFit[Link to code](#)

Wraps the executable 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
'dtiffit -k diffusion.nii -o TP -m mask.nii -r bvecs -b bvals'

```

Inputs:

```

[Mandatory]
mask: (an existing file name)
      bet binary mask file
      argument: ``-m %s``, position: 2
dwi: (an existing file name)
      diffusion weighted image data file
      argument: ``-k %s``, position: 0
bvecs: (an existing file name)
        b vectors file
        argument: ``-r %s``, position: 3
bvals: (an existing file name)
        b values file
        argument: ``-b %s``, position: 4

[Optional]
little_bit: (a boolean)
            only process small area of brain
            argument: ``--littlebit``
cni: (an existing file name)
      input counfound regressors
      argument: ``--cni=%s``
max_x: (an integer (int or long))
        max x
        argument: ``-X %d``
max_z: (an integer (int or long))
        max z
        argument: ``-Z %d``
min_x: (an integer (int or long))
        min x
        argument: ``-x %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
sse: (a boolean)
      output sum of squared errors
      argument: ``--sse``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
              FSL output type
min_z: (an integer (int or long))
        min z
        argument: ``-z %d``
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

        Additional parameters to the command
        argument: ``%s``
base_name: (a unicode string, nipyype default value: dtifit_)
        base_name that all output files will start with
        argument: ``-o %s``, position: 1
max_y: (an integer (int or long))
        max y
        argument: ``-Y %d``
gradnonlin: (an existing file name)
        gradient non linearities
        argument: ``--gradnonlin=%s``
min_y: (an integer (int or long))
        min y
        argument: ``-y %d``
save_tensor: (a boolean)
        save the elements of the tensor
        argument: ``--save_tensor``

```

Outputs:

```

V2: (an existing file name)
    path/name of file with the 2nd eigenvector
L3: (an existing file name)
    path/name of file with the 3rd eigenvalue
FA: (an existing file name)
    path/name of file with the fractional 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
M0: (an existing file name)
    path/name of file with the mode of anisotropy
tensor: (an existing file name)
    path/name of file with the 4D tensor volume
sse: (an existing file name)
    path/name of file with the summed squared error
L1: (an existing file name)
    path/name of file with the 1st eigenvalue
MD: (an existing file name)
    path/name of file with the mean diffusivity
L2: (an existing file name)
    path/name of file with the 2nd eigenvalue
V3: (an existing file name)
    path/name of file with the 3rd eigenvector

```

References:

None

63.2.3 DistanceMap[Link to code](#)Wraps the executable command `distancemap`.Use FSL's `distancemap` to generate a map of the distance to the nearest nonzero voxel.

Example

```
>>> import nipyype.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
        argument: ``--in=%s``

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
local_max_file: (a boolean or a file name)
                write an image of the local maxima
                argument: ``--localmax=%s``
distance_map: (a file name)
               distance map to write
               argument: ``--out=%s``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
mask_file: (an existing file name)
            binary mask to constrain calculations
            argument: ``--mask=%s``
invert_input: (a boolean)
              invert input image
              argument: ``--invert``
environ: (a dictionary with keys which are a 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
```

Outputs:

```
local_max_file: (a file name)
                image of local maxima
distance_map: (an existing file name)
              value is distance to nearest nonzero voxels
```

References:

None

63.2.4 FindTheBiggest

[Link to code](#)

Wraps the executable command `find_the_biggest`.

Use FSL `find_the_biggest` for performing hard segmentation on the outputs of connectivity-based thresholding in `protrack`. 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
          argument: ``%s``, position: 0

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
out_file: (a file name)
          file with the resulting segmentation
          argument: ``%s``, position: 2
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
```

Outputs:

```
out_file: (an existing file name)
          output file indexed in order of input files
          argument: ``%s``
```

References:

None

63.2.5 MakeDyadicVectors

[Link to code](#)

Wraps the executable command `make_dyadic_vectors`.

Create vector volume representing mean principal diffusion direction and its uncertainty (dispersion)

Inputs:

```
[Mandatory]
phi_vol: (an existing file name)
          argument: ``%s``, position: 1
theta_vol: (an existing file name)
           argument: ``%s``, position: 0

[Optional]
mask: (an existing file name)
      argument: ``%s``, position: 2
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
```

(continues on next page)

(continued from previous page)

```

        FSL output type
output: (a file name, nipy default value: dyads)
        argument: ``%s``, position: 3
perc: (a float)
        the {perc}% angle of the output cone of uncertainty (output will be
        in degrees)
        argument: ``%f``, position: 4
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

dispersion: (an existing file name)
dyads: (an existing file name)

```

References:

None

63.2.6 ProbTrackX[Link to code](#)

Wraps the executable command probtrackx.

Use FSL probtrackx for tractography on bedpostx results

Examples

```

>>> from nipy.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]
mask: (an existing file name)
        bet binary mask file in diffusion space
        argument: ``-m %s``
thsamples: (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
        argument: ``--seed=%s``

```

(continues on next page)

(continued from previous page)

```

phsamples: (a list of items which are an existing file name)
fsamples: (a list of items which are an existing file name)

[Optional]
fibst: (an integer (int or long))
    force a starting fibre for tracking - default=1, i.e. first fibre
    orientation. Only works if randfib==0
    argument: ``--fibst=%d``
s2tastext: (a boolean)
    output seed-to-target counts as a text file (useful when seeding
    from a mesh)
    argument: ``--s2tastext``
verbose: (0 or 1 or 2)
    Verbose level, [0-2]. Level 2 is required to output particle files.
    argument: ``--verbose=%d``
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)
    argument: ``--randfib=%d``
loop_check: (a boolean)
    perform loop_checks on paths - slower, but allows lower curvature
    threshold
    argument: ``--loopcheck``
stop_mask: (an existing file name)
    stop tracking at locations given by this mask file
    argument: ``--stop=%s``
use_anisotropy: (a boolean)
    use anisotropy to constrain tracking
    argument: ``--usef``
dist_thresh: (a float)
    discards samples shorter than this threshold (in mm - default=0)
    argument: ``--distthresh=%.3f``
xfm: (an existing file name)
    transformation matrix taking seed space to DTI space (either FLIRT
    matrix or FNIRT warp_field) - default is identity
    argument: ``--xfm=%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
mode: ('simple' or 'two_mask_symm' or 'seedmask')
    options: simple (single seed voxel), seedmask (mask of seed voxels),
    twomask_symm (two bet binary masks)
    argument: ``--mode=%s``
out_dir: (an existing directory name)
    directory to put the final volumes in
    argument: ``--dir=%s``
correct_path_distribution: (a boolean)
    correct path distribution for the length of the pathways
    argument: ``--pd``
c_thresh: (a float)
    curvature threshold - default=0.2
    argument: ``--cthr=%.3f``
sample_random_points: (a boolean)
    sample random points within seed voxels
    argument: ``--sampvox``

```

(continues on next page)

(continued from previous page)

```

network: (a boolean)
    activate network mode - only keep paths going through at least one
    seed mask (required if multiple seed masks)
    argument: ``--network``
opd: (a boolean, nipy default value: True)
    outputs path distributions
    argument: ``--opd``
force_dir: (a boolean, nipy default value: True)
    use the actual directory name given - i.e. do not add + to make a
    new directory
    argument: ``--forcedir``
n_samples: (an integer (int or long), nipy default value: 5000)
    number of samples - default=5000
    argument: ``--nsamples=%d``
mod_euler: (a boolean)
    use modified euler streamlining
    argument: ``--modeuler``
step_length: (a float)
    step_length in mm - default=0.5
    argument: ``--steplength=%.3f``
avoid_mp: (an existing file name)
    reject pathways passing through locations given by this mask
    argument: ``--avoid=%s``
n_steps: (an integer (int or long))
    number of steps per sample - default=2000
    argument: ``--nsteps=%d``
os2t: (a boolean)
    Outputs seeds to targets
    argument: ``--os2t``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
inv_xfm: (a file name)
    transformation matrix taking DTI space to seed space (compulsory
    when using a warp_field for seeds_to_dti)
    argument: ``--invxfm=%s``
mask2: (an existing file name)
    second bet binary mask (in diffusion space) in twomask_symm mode
    argument: ``--mask2=%s``
seed_ref: (an existing file name)
    reference vol to define seed space in simple mode - diffusion space
    assumed if absent
    argument: ``--seedref=%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
waypoints: (an existing file name)
    waypoint mask or ascii list of waypoint masks - only keep paths
    going through ALL the masks
    argument: ``--waypoints=%s``
mesh: (an existing file name)
    Freesurfer-type surface descriptor (in ascii format)
    argument: ``--mesh=%s``
target_masks: (a list of items which are a file name)
    list of target masks - required for seeds_to_targets classification
    argument: ``--targetmasks=%s``
random_seed: (a boolean)

```

(continues on next page)

(continued from previous page)

```

    random seed
    argument: ``--rseed``
samples_base_name: (a unicode string, nipy default value: merged)
    the rootname/base_name for samples files
    argument: ``--samples=%s``

```

Outputs:

```

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
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
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

```

References:

None

63.2.7 ProbTrackX2[Link to code](#)

Wraps the executable 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]
mask: (an existing file name)
    bet binary mask file in diffusion space
    argument: ``-m %s``
thsamples: (a list of items which are an existing file name)

```

(continues on next page)

(continued from previous page)

```

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
      argument: ``--seed=%s``
phsamples: (a list of items which are an existing file name)
fsamples: (a list of items which are an existing file name)

[Optional]
fibst: (an integer (int or long))
      force a starting fibre for tracking - default=1, i.e. first fibre
      orientation. Only works if randfib==0
      argument: ``--fibst=%d``
lrtarget3: (an existing file name)
      Column-space mask used for NxN connectivity matrix
      argument: ``--lrtarget3=%s``
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)
      argument: ``--randfib=%d``
stop_mask: (an existing file name)
      stop tracking at locations given by this mask file
      argument: ``--stop=%s``
colmask4: (an existing file name)
      Mask for columns of matrix4 (default=seed mask)
      argument: ``--colmask4=%s``
dist_thresh: (a float)
      discards samples shorter than this threshold (in mm - default=0)
      argument: ``--distthresh=%.3f``
simple: (a boolean)
      rack from a list of voxels (seed must be a ASCII list of
      coordinates)
      argument: ``--simple``
out_dir: (an existing directory name)
      directory to put the final volumes in
      argument: ``--dir=%s``
meshspace: ('caret' or 'freesurfer' or 'first' or 'vox')
      Mesh reference space - either "caret" (default) or "freesurfer" or
      "first" or "vox"
      argument: ``--meshspace=%s``
omatrix2: (a boolean)
      Output matrix2 - SeedToLowResMask
      argument: ``--omatrix2``
      requires: target2
distthresh3: (a float)
      Discards samples (in matrix3) shorter than this threshold (in mm -
      default=0)
      argument: ``--distthresh3=%.3f``
fopd: (an existing file name)
      Other mask for binning tract distribution
      argument: ``--fopd=%s``
sample_random_points: (a boolean)
      sample random points within seed voxels
      argument: ``--sampvox``
network: (a boolean)
      activate network mode - only keep paths going through at least one

```

(continues on next page)

(continued from previous page)

```

        seed mask (required if multiple seed masks)
        argument: ``--network``
n_samples: (an integer (int or long), nipy default value: 5000)
        number of samples - default=5000
        argument: ``--nsamples=%d``
target4: (an existing file name)
        Brain mask in DTI space
        argument: ``--target4=%s``
avoid_mp: (an existing file name)
        reject pathways passing through locations given by this mask
        argument: ``--avoid=%s``
n_steps: (an integer (int or long))
        number of steps per sample - default=2000
        argument: ``--nsteps=%d``
os2t: (a boolean)
        Outputs seeds to targets
        argument: ``--os2t``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_GZ')
        FSL output type
inv_xfm: (a file name)
        transformation matrix taking DTI space to seed space (compulsory
        when using a warp_field for seeds_to_dti)
        argument: ``--invxfm=%s``
seed_ref: (an existing file name)
        reference vol to define seed space in simple mode - diffusion space
        assumed if absent
        argument: ``--seedref=%s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
loop_check: (a boolean)
        perform loop_checks on paths - slower, but allows lower curvature
        threshold
        argument: ``--loopcheck``
verbose: (0 or 1 or 2)
        Verbose level, [0-2]. Level 2 is required to output particle files.
        argument: ``--verbose=%d``
random_seed: (a boolean)
        random seed
        argument: ``--rseed``
samples_base_name: (a unicode string, nipy default value: merged)
        the rootname/base_name for samples files
        argument: ``--samples=%s``
waycond: ('OR' or 'AND')
        Waypoint condition. Either "AND" (default) or "OR"
        argument: ``--waycond=%s``
omatrix3: (a boolean)
        Output matrix3 (NxN connectivity matrix)
        argument: ``--omatrix3``
        requires: target3, lrtarget3
target2: (an existing file name)
        Low resolution binary brain mask for storing connectivity
        distribution in matrix2 mode
        argument: ``--target2=%s``
waypoints: (an existing file name)
        waypoint mask or ascii list of waypoint masks - only keep paths

```

(continues on next page)

(continued from previous page)

```

    going through ALL the masks
    argument: ``--waypoints=%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
wayorder: (a boolean)
    Reject streamlines that do not hit waypoints in given order. Only
    valid if waycond=AND
    argument: ``--wayorder``
xfm: (an existing file name)
    transformation matrix taking seed space to DTI space (either FLIRT
    matrix or FNIRT warp_field) - default is identity
    argument: ``--xfm=%s``
onewaycondition: (a boolean)
    Apply waypoint conditions to each half tract separately
    argument: ``--onewaycondition``
correct_path_distribution: (a boolean)
    correct path distribution for the length of the pathways
    argument: ``--pd``
use_anisotropy: (a boolean)
    use anisotropy to constrain tracking
    argument: ``--usef``
distthresh1: (a float)
    Discards samples (in matrix1) shorter than this threshold (in mm -
    default=0)
    argument: ``--distthresh1=%.3f``
omatrix4: (a boolean)
    Output matrix4 - DtiMaskToSeed (special Oxford Sparse Format)
    argument: ``--omatrix4``
force_dir: (a boolean, nipy default value: True)
    use the actual directory name given - i.e. do not add + to make a
    new directory
    argument: ``--forcedir``
target_masks: (a list of items which are a file name)
    list of target masks - required for seeds_to_targets classification
    argument: ``--targetmasks=%s``
s2tastext: (a boolean)
    output seed-to-target counts as a text file (useful when seeding
    from a mesh)
    argument: ``--s2tastext``
step_length: (a float)
    step_length in mm - default=0.5
    argument: ``--steplength=%.3f``
opd: (a boolean, nipy default value: True)
    outputs path distributions
    argument: ``--opd``
c_thresh: (a float)
    curvature threshold - default=0.2
    argument: ``--cthr=%.3f``
target3: (an existing file name)
    Mask used for NxN connectivity matrix (or NxN if lrtarget3 is set)
    argument: ``--target3=%s``
omatrix1: (a boolean)
    Output matrix1 - SeedToSeed Connectivity
    argument: ``--omatrix1``
mod_euler: (a boolean)

```

(continues on next page)

(continued from previous page)

```
use modified euler streamlining
argument: ``--modeuler``
```

Outputs:

```
lookup_tractspace: (an existing file name)
    lookup_tractspace generated by --omatrix2 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
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
network_matrix: (an existing file name)
    the network matrix generated by --omatrix1 option
matrix2_dot: (an existing file name)
    Output matrix2.dot - SeedToLowResMask
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
matrix1_dot: (an existing file name)
    Output matrix1.dot - SeedToSeed Connectivity
matrix3_dot: (an existing file name)
    Output matrix3 - NxN connectivity matrix
targets: (a list of items which are an existing file name)
    a list with all generated seeds_to_target files
log: (an existing file name)
    path/name of a text record of the command that was run
```

References:

None

63.2.8 ProjThresh[Link to code](#)Wraps the executable 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 nipy.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
    argument: ``%s``, position: 0
threshold: (an integer (int or long))
```

(continues on next page)

(continued from previous page)

```

        threshold indicating minimum number of seed voxels entering this
        mask region
        argument: ``%d``, position: 1

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
        Environment variables
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``

```

Outputs:

```

out_files: (a list of items which are an existing file name)
           path/name of output volume after thresholding

```

References:

None

63.2.9 TractSkeleton[Link to code](#)Wraps the executable 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 (typically mean FA volume)
        argument: ``-i %s``

[Optional]
use_cingulum_mask: (a boolean, nipy default value: True)
                  perform alternate search using built-in cingulum mask

```

(continues on next page)

(continued from previous page)

```

        mutually_exclusive: search_mask_file
skeleton_file: (a boolean or a file name)
    write out skeleton image
    argument: ``-o %s``
projected_data: (a file name)
    input data projected onto skeleton
alt_skeleton: (an existing file name)
    alternate skeleton to use
    argument: ``-s %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
alt_data_file: (an existing file name)
    4D non-FA data to project onto skeleton
    argument: ``-a %s``
search_mask_file: (an existing file name)
    mask in which to use alternate search rule
    mutually_exclusive: use_cingulum_mask
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
distance_map: (an existing file name)
    distance map image
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
threshold: (a float)
    skeleton threshold value
project_data: (a boolean)
    project data onto skeleton
    argument: ``-p %.3f %s %s %s %s``
    requires: threshold, distance_map, data_file
data_file: (an existing file name)
    4D data to project onto skeleton (usually FA)

```

Outputs:

```

skeleton_file: (a file name)
    tract skeleton image
projected_data: (a file name)
    input data projected onto skeleton

```

References:

None

63.2.10 VecReg[Link to code](#)

Wraps the executable 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 nipy.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
        argument: ``-i %s``
ref_vol: (an existing file name)
        filename for reference (target) volume
        argument: ``-r %s``

[Optional]
mask: (an existing file name)
      brain mask in input space
      argument: ``-m %s``
interpolation: ('nearestneighbour' or 'trilinear' or 'sinc' or
               'spline')
               interpolation method : nearestneighbour, trilinear (default), sinc
               or spline
               argument: ``--interp=%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
              argument: ``--rotmat=%s``
warp_field: (an existing file name)
            filename for 4D warp field for nonlinear registration
            argument: ``-w %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
             'NIFTI_GZ')
             FSL output type
affine_mat: (an existing file name)
            filename for affine transformation matrix
            argument: ``-t %s``
ref_mask: (an existing file name)
          brain mask in output space (useful for speed up of nonlinear reg)
          argument: ``--refmask=%s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%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
              argument: ``--rotwarp=%s``
out_file: (a file name)
          filename for output registered vector or tensor field
          argument: ``-o %s``
```


Outputs:

```
out_file: (an existing file name)
          path/name of filename for the registered vector or tensor field
```

References:

None

63.2.11 XFibres5[Link to code](#)Wraps the executable command `xfibres`.

Perform model parameters estimation for local (voxelwise) diffusion parameters

Inputs:

```
[Mandatory]
mask: (an existing file name)
      brain binary mask file (i.e. from BET)
      argument: ``--mask=%s``
dwi: (an existing file name)
      diffusion weighted image data file
      argument: ``--data=%s``
bvecs: (an existing file name)
        b vectors file
        argument: ``--bvecs=%s``
n_fibres: (a long integer >= 1, nipy default value: 2)
          Maximum number of fibres to fit in each voxel
          argument: ``--nfibres=%d``
bvals: (an existing file name)
        b values file
        argument: ``--bvals=%s``

[Optional]
n_jumps: (an integer (int or long), nipy default value: 5000)
          Num of jumps to be made by MCMC
          argument: ``--njumps=%d``
f0_ard: (a boolean)
          Noise floor model: add to the model an unattenuated signal
          compartment f0
          argument: ``--f0 --ardf0``
          mutually_exclusive: f0_noard, f0_ard, all_ard
logdir: (a directory name, nipy default value: .)
          argument: ``--logdir=%s``
burn_in: (a long integer >= 0, nipy default value: 0)
          Total num of jumps at start of MCMC to be discarded
          argument: ``--burnin=%d``
fudge: (an integer (int or long))
        ARD fudge factor
        argument: ``--fudge=%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
f0_noard: (a boolean)
          Noise floor model: add to the model an unattenuated signal
          compartment f0
```

(continues on next page)

(continued from previous page)

```

        argument: ``--f0``
        mutually_exclusive: f0_noard, f0_ard
seed: (an integer (int or long))
    seed for pseudo random number generator
    argument: ``--seed=%d``
no_spat: (a boolean)
    Initialise with tensor, not spatially
    argument: ``--nospat``
    mutually_exclusive: no_spat, non_linear, clinear
no_ard: (a boolean)
    Turn ARD off on all fibres
    argument: ``--noard``
    mutually_exclusive: no_ard, all_ard
force_dir: (a boolean, nipy default value: True)
    use the actual directory name given (do not add + to make a new
    directory)
    argument: ``--forcedir``
sample_every: (a long integer >= 0, nipy default value: 1)
    Num of jumps for each sample (MCMC)
    argument: ``--sampleevery=%d``
non_linear: (a boolean)
    Initialise with nonlinear fitting
    argument: ``--nonlinear``
    mutually_exclusive: no_spat, non_linear, clinear
burn_in_no_ard: (a long integer >= 0, nipy default value: 0)
    num of burnin jumps before the ard is imposed
    argument: ``--burnin_noard=%d``
update_proposal_every: (a long integer >= 1, nipy default value:
    40)
    Num of jumps for each update to the proposal density std (MCMC)
    argument: ``--updateproposalevery=%d``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
clinear: (a boolean)
    Initialise with constrained nonlinear fitting
    argument: ``--cnonlinear``
    mutually_exclusive: no_spat, non_linear, clinear
rician: (a boolean)
    use Rician noise modeling
    argument: ``--rician``
model: (1 or 2 or 3)
    use monoexponential (1, default, required for single-shell) or
    multiexponential (2, multi-shell) model
    argument: ``--model=%d``
gradnonlin: (an existing file name)
    gradient file corresponding to slice
    argument: ``--gradnonlin=%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
all_ard: (a boolean)
    Turn ARD on on all fibres
    argument: ``--allard``
    mutually_exclusive: no_ard, all_ard

```

Outputs:

```

thsamples: (a list of items which are an existing file name)
            theta samples, per fiber
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
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_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

```

References:

None

63.3 interfaces.fsl.epi

63.3.1 ApplyTOPUP

[Link to code](#)

Wraps the executable 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]
in_files: (a list of items which are an existing file name)
           name of file with images
           argument: ``--imain=%s``
encoding_file: (an existing file name)
               name of text file with PE directions/times
               argument: ``--datain=%s``

[Optional]

```

(continues on next page)

(continued from previous page)

```

in_topup_movpar: (an existing file name)
    topup movpar.txt file
    requires: in_topup_fieldcoef
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
datatype: ('char' or 'short' or 'int' or 'float' or 'double')
    force output data type
    argument: ``-d=%s``
in_topup_fieldcoef: (an existing file name)
    topup file containing the field coefficients
    argument: ``--topup=%s``
    requires: in_topup_movpar
interp: ('trilinear' or 'spline')
    interpolation method
    argument: ``--interp=%s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
out_corrected: (a file name)
    output (warped) image
    argument: ``--out=%s``
method: ('jac' or 'lsr')
    use jacobian modulation (jac) or least-squares resampling (lsr)
    argument: ``--method=%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
in_index: (a list of items which are an integer (int or long))
    comma separated list of indices corresponding to --datain
    argument: ``--inindex=%s``

```

Outputs:

```

out_corrected: (an existing file name)
    name of 4D image file with unwarped images

```

References:

None

63.3.2 EPIDeWarp[Link to code](#)Wraps the executable 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
    argument: ``--dph %s``
mag_file: (an existing file name)
    Magnitude file
    argument: ``--mag %s``, position: 0

[Optional]
tmpdir: (a string)
    tmpdir
    argument: ``--tmpdir %s``
cleanup: (a boolean)
    cleanup
    argument: ``--cleanup``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
epi_file: (an existing file name)
    EPI volume to unwarp
    argument: ``--epi %s``
exf_file: (an existing file name)
    example func volume (or use epi)
    argument: ``--exf %s``
exfdw: (a string)
    dewarped example func volume
    argument: ``--exfdw %s``
sigma: (an integer (int or long), nipy default value: 2)
    2D spatial gaussian smoothing stdev (default = 2mm)
    argument: ``--sigma %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
tediff: (a float, nipy default value: 2.46)
    difference in B0 field map TEs
    argument: ``--tediff %s``
esp: (a float, nipy default value: 0.58)
    EPI echo spacing
    argument: ``--esp %s``
vsm: (a string)
    voxel shift map
    argument: ``--vsm %s``
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

        Additional parameters to the command
        argument: ``%s``
epidw: (a string)
        dewarped epi volume
        argument: ``--epidw %s``
nocleanup: (a boolean, nipy default value: True)
        no cleanup
        argument: ``--nocleanup``

```

Outputs:

```

exfdw: (a file name)
        dewarped functional volume example
exf_mask: (a file name)
        Mask from example functional volume
unwarped_file: (a file name)
        unwarped epi file
vsm_file: (a file name)
        voxel shift map

```

References:

None

63.3.3 Eddy[Link to code](#)

Wraps the executable 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 nipy.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 --ff=10.0 --acqp=epi_acqp.txt --bvals=bvals.scheme --bvecs=bvecs.
↪scheme --imain=epi.nii --index=epi_index.txt --mask=epi_mask.nii --niter=5 --
↪nvoxhp=1000 --out=.../eddy_corrected'
>>> eddy.inputs.use_cuda = False
>>> eddy.cmdline
'eddy_openmp --ff=10.0 --acqp=epi_acqp.txt --bvals=bvals.scheme --bvecs=bvecs.
↪scheme --imain=epi.nii --index=epi_index.txt --mask=epi_mask.nii --niter=5 --
↪nvoxhp=1000 --out=.../eddy_corrected'
>>> res = eddy.run()

```

Inputs:

```

[Mandatory]
in_bval: (an existing file name)
    File containing the b-values for all volumes in --imain
    argument: ``--bvals=%s``
in_bvec: (an existing file name)
    File containing the b-vectors for all volumes in --imain
    argument: ``--bvecs=%s``
in_acqp: (an existing file name)
    File containing acquisition parameters
    argument: ``--acqp=%s``
in_file: (an existing file name)
    File containing all the images to estimate distortions for
    argument: ``--imain=%s``
in_mask: (an existing file name)
    Mask to indicate brain
    argument: ``--mask=%s``
in_index: (an existing file name)
    File containing indices for all volumes in --imain into --acqp and
    --topup
    argument: ``--index=%s``

[Optional]
in_topup_movpar: (an existing file name)
    topup movpar.txt file
    requires: in_topup_fieldcoef
use_cuda: (a boolean)
    Run eddy using cuda gpu
flm: ('linear' or 'quadratic' or 'cubic')
    First level EC model
    argument: ``--flm=%s``
out_base: (a unicode string, nipy default value: eddy_corrected)
    basename for output (warped) image
    argument: ``--out=%s``
niter: (an integer (int or long), nipy default value: 5)
    Number of iterations
    argument: ``--niter=%s``
is_shelled: (a boolean)
    Override internal check to ensure that data are acquired on a set of
    b-value shells
    argument: ``--data_is_shelled``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
residuals: (a boolean)
    Output Residuals
    argument: ``--residuals``
interp: ('spline' or 'trilinear')
    Interpolation model for estimation step
    argument: ``--interp=%s``
session: (an existing file name)
    File containing session indices for all volumes in --imain
    argument: ``--session=%s``
in_topup_fieldcoef: (an existing file name)
    topup file containing the field coefficients
    argument: ``--topup=%s``
    requires: in_topup_movpar

```

(continues on next page)

(continued from previous page)

```

fudge_factor: (a float, nipy default value: 10.0)
    Fudge factor for hyperparameter error variance
    argument: ``--ff=%s``
fep: (a boolean)
    Fill empty planes in x- or y-directions
    argument: ``--fep``
slm: ('none' or 'linear' or 'quadratic')
    Second level EC model
    argument: ``--slm=%s``
dont_sep_offs_move: (a boolean)
    Do NOT attempt to separate field offset from subject movement
    argument: ``--dont_sep_offs_move``
repol: (a boolean)
    Detect and replace outlier slices
    argument: ``--repol``
num_threads: (an integer (int or long), nipy default value: 1)
    Number of openmp threads to use
nvoxhp: (an integer (int or long), nipy default value: 1000)
    # of voxels used to estimate the hyperparameters
    argument: ``--nvoxhp=%s``
dont_peas: (a boolean)
    Do NOT perform a post-eddy alignment of shells
    argument: ``--dont_peas``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
field: (a unicode string)
    NonTOPUP fieldmap scaled in Hz - filename has to be provided without
    an extension. TOPUP is strongly recommended
    argument: ``--field=%s``
method: ('jac' or 'lsr')
    Final resampling method (jacobian/least squares)
    argument: ``--resamp=%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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
    argument: ``--field_mat=%s``
cnr_maps: (a boolean)
    Output CNR-Maps
    argument: ``--cnr_maps``
fwhm: (a float)
    FWHM for conditioning filter when estimating the parameters
    argument: ``--fwhm=%s``

```

Outputs:

```

out_movement_rms: (an existing file name)
    Summary of the "total movement" in each volume
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
out_residuals: (an existing file name)
    path/name of file with the residuals

```

(continues on next page)

(continued from previous page)

```

out_corrected: (an existing file name)
    4D image file containing all the corrected volumes
out_restricted_movement_rms: (an existing file name)
    Summary of the "total movement" in each volume disregarding
    translation in the PE direction
out_cnr_maps: (an existing file name)
    path/name of file with the cnr_maps
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_rotated_bvecs: (an existing file name)
    File containing rotated b-values for all volumes

```

References:

None

63.3.4 EddyCorrect[Link to code](#)Wraps the executable 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
    argument: ``%s``, position: 0
ref_num: (an integer (int or long), nipy default value: 0)
    reference number
    argument: ``%d``, position: 2

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
out_file: (a file name)
    4D output file
    argument: ``%s``, position: 1
args: (a unicode string)
    Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
Environment variables

```

Outputs:

```

eddy_corrected: (an existing file name)
                path/name of 4D eddy corrected output file

```

References:

None

63.3.5 EpiReg[Link to code](#)Wraps the executable command `epi_reg`.Runs FSL `epi_reg` script for simultaneous coregistration and fieldmap unwarping.**Examples**

```

>>> from nipy.interfaces.fsl import EpiReg
>>> epiрег = EpiReg()
>>> epiрег.inputs.epi='epi.nii'
>>> epiрег.inputs.t1_head='T1.nii'
>>> epiрег.inputs.t1_brain='T1_brain.nii'
>>> epiрег.inputs.out_base='epi2struct'
>>> epiрег.inputs.fmap='fieldmap_phase_fslprepared.nii'
>>> epiрег.inputs.fmapmag='fieldmap_mag.nii'
>>> epiрег.inputs.fmapmagbrain='fieldmap_mag_brain.nii'
>>> epiрег.inputs.echospacing=0.00067
>>> epiрег.inputs.pedir='y'
>>> epiрег.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'
>>> epiрег.run()

```

Inputs:

```

[Mandatory]
t1_head: (an existing file name)
         wholehead T1 image
         argument: ``--t1=%s``, position: -3
t1_brain: (an existing file name)
          brain extracted T1 image
          argument: ``--t1brain=%s``, position: -2
epi: (an existing file name)
     EPI image
     argument: ``--epi=%s``, position: -4

[Optional]
echospacing: (a float)
             Effective EPI echo spacing (sometimes called dwell time) - in

```

(continues on next page)

(continued from previous page)

```

seconds
argument: ``--echospadding=%f``
no_clean: (a boolean, nipy default value: True)
do not clean up intermediate files
argument: ``--noclean``
out_base: (a string, nipy default value: epi2struct)
output base name
argument: ``--out=%s``, position: -1
weight_image: (an existing file name)
weighting image (in T1 space)
argument: ``--weight=%s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipy default value: {})
Environment variables
no_fmmapreg: (a boolean)
do not perform registration of fmap to T1 (use if fmap already
registered)
argument: ``--nofmmapreg``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
'NIFTI_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
argument: ``--pedir=%s``
fmmapmag: (an existing file name)
fieldmap magnitude image - wholehead
argument: ``--fmmapmag=%s``
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
fmap: (an existing file name)
fieldmap image (in rad/s)
argument: ``--fmap=%s``
fmmapmagbrain: (an existing file name)
fieldmap magnitude image - brain extracted
argument: ``--fmmapmagbrain=%s``
wmseg: (an existing file name)
white matter segmentation of T1 image, has to be named like the
tlbrain and end on _wmseg
argument: ``--wmseg=%s``

```

Outputs:

```

fullwarp: (an existing file name)
warpfield to unwarp epi and transform into structural space
out_1vol: (an existing file name)
unwarped and coregistered single volume
shiftmap: (an existing file name)
shiftmap in epi space
epi2str_mat: (an existing file name)
rigid epi-to-structural transform
seg: (an existing file name)
white matter, gray matter, csf segmentation
fmap2str_mat: (an existing file name)
rigid fieldmap-to-structural transform
epi2str_inv: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

        rigid structural-to-epi transform
fmap2epi_mat: (an existing file name)
        rigid fieldmap-to-epi transform
fmapmag_str: (an existing file name)
        fieldmap magnitude image in structural space
fmap_str: (an existing file name)
        fieldmap in structural space
fmap_epi: (an existing file name)
        fieldmap in epi space
wmedge: (an existing file name)
        white matter edges for visualization
out_file: (an existing file name)
        unwarped and coregistered epi input
wmseg: (an existing file name)
        white matter segmentation used in flirt bbr

```

References:

None

63.3.6 PrepareFieldmap[Link to code](#)Wraps the executable 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)
        argument: ``%f``, position: -2
in_magnitude: (an existing file name)
        Magnitude difference map, brain extracted
        argument: ``%s``, position: 3
in_phase: (an existing file name)
        Phase difference map, in SIEMENS format range from 0-4096 or 0-8192)
        argument: ``%s``, position: 2

```

(continues on next page)

(continued from previous page)

```
[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
              FSL output type
out_fieldmap: (a file name)
              output name for prepared fieldmap
              argument: ``%s``, position: 4
nocheck: (a boolean, nipy default value: False)
          do not perform sanity checks for image size/range/dimensions
          argument: ``--nocheck``, position: -1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
scanner: (a string, nipy default value: SIEMENS)
         must be SIEMENS
         argument: ``%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
```

Outputs:

```
out_fieldmap: (an existing file name)
              output name for prepared fieldmap
```

References:

None

63.3.7 SigLoss[Link to code](#)

Wraps the executable 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
         argument: ``-i %s``

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
```

(continues on next page)

(continued from previous page)

```

        'NIFTI_GZ')
        FSL output type
mask_file: (an existing file name)
        brain mask file
        argument: ``-m %s``
echo_time: (a float)
        echo time in seconds
        argument: ``--te=%f``
out_file: (a file name)
        output signal loss estimate file
        argument: ``-s %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
slice_direction: ('x' or 'y' or 'z')
        slicing direction
        argument: ``-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

```

Outputs:

```

out_file: (an existing file name)
        signal loss estimate file

```

References:

None

63.3.8 TOPUP[Link to code](#)Wraps the executable 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 nipy.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 --rbmout=xfm --dfout=warpfield'
>>> res = topup.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        name of 4D file with images

```

(continues on next page)

(continued from previous page)

```

        argument: ``--imain=%s``
encoding_file: (an existing file name)
    name of text file with PE directions/times
    argument: ``--datain=%s``
    mutually_exclusive: encoding_direction
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
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
    argument: ``--datain=%s``
    mutually_exclusive: encoding_file
    requires: readout_times

[Optional]
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.
    argument: ``--ssqlambda=%d``
numprec: ('double' or 'float')
    Precision for representing Hessian, double or float.
    argument: ``--numprec=%s``
config: (a string, nipype default value: b02b0.cnf)
    Name of config file specifying command line arguments
    argument: ``--config=%s``
out_base: (a file name)
    base-name of output files (spline coefficients (Hz) and movement
    parameters)
    argument: ``--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', nipype default value: {})
    Environment variables
splineorder: (an integer (int or long))
    order of spline, 2->Qadratic spline, 3->Cubic spline
    argument: ``--splineorder=%d``
out_warp_prefix: (a unicode string, nipype default value: warpfield)
    prefix for the warpfield images (in mm)
    argument: ``--dfout=%s``
out_jac_prefix: (a unicode string, nipype default value: jac)
    prefix for the warpfield images
    argument: ``--jacout=%s``
warp_res: (a float)
    (approximate) resolution (in mm) of warp basis for the different
    sub-sampling levels
    argument: ``--warpres=%f``
interp: ('spline' or 'linear')
    Image interpolation model, linear or spline.
    argument: ``--interp=%s``
subsamp: (an integer (int or long))
    sub-sampling scheme
    argument: ``--subsamp=%d``

```

(continues on next page)

(continued from previous page)

```

regrid: (1 or 0)
    If set (=1), the calculations are done in a different grid
    argument: ``--regrid=%d``
max_iter: (an integer (int or long))
    max # of non-linear iterations
    argument: ``--miter=%d``
out_logfile: (a file name)
    name of log-file
    argument: ``--logout=%s``
minmet: (0 or 1)
    Minimisation method 0=Levenberg-Marquardt, 1=Scaled Conjugate
    Gradient
    argument: ``--minmet=%d``
reg_lambda: (a float)
    Weight of regularisation, default depending on --ssqlambda and
    --regmod switches.
    argument: ``--lambda=%0.f``
out_mat_prefix: (a unicode string, nipy default value: xfm)
    prefix for the realignment matrices
    argument: ``--rbmout=%s``
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.
    argument: ``--regmod=%s``
scale: (0 or 1)
    If set (=1), the images are individually scaled to a common mean
    argument: ``--scale=%d``
out_field: (a file name)
    name of image file with field (Hz)
    argument: ``--fout=%s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
out_corrected: (a file name)
    name of 4D image file with unwarped images
    argument: ``--iout=%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
fwhm: (a float)
    FWHM (in mm) of gaussian smoothing kernel
    argument: ``--fwhm=%f``
estmov: (1 or 0)
    estimate movements if set
    argument: ``--estmov=%d``

```

Outputs:

```

out_movpar: (an existing file name)
    movpar.txt output file
out_fieldcoef: (an existing file name)
    file containing the field coefficients

```

(continues on next page)

(continued from previous page)

```

out_corrected: (a file name)
    name of 4D image file with unwarped images
out_logfile: (a file name)
    name of log-file
out_mats: (a list of items which are an existing file name)
    realignment matrices
out_warps: (a list of items which are an existing file name)
    warpfild images
out_jacs: (a list of items which are an existing file name)
    Jacobian 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)

```

References:

None

63.4 interfaces.fsl.fix**63.4.1 Classifier**[Link to code](#)

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.
    argument: ``%d``, position: -1
trained_wts_file: (an existing file name)
    trained-weights file
    argument: ``%s``, position: 2

[Optional]
artifacts_list_file: (a file name)
    Text file listing which ICs are artifacts; can be the output from
    classification or can be created manually
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a 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
mel_ica: (an existing directory name)
    Melodic output directory or directories
    argument: ``%s``, position: 1

```

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

```

63.4.2 Cleaner

[Link to code](#)

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
    argument: ``%s``, position: 1

[Optional]
confound_file: (a file name)
    Include additional confound file.
    argument: ``-x %s``, position: 4
confound_file_2: (a file name)
    Include additional confound file.
    argument: ``-x %s``, position: 6
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
cleanup_motion: (a boolean)
    cleanup motion confounds, looks for design.fsf for highpass filter
    cut-off
    argument: ``-m``, position: 2
aggressive: (a boolean)
    Apply aggressive (full variance) cleanup, instead of the default
    less-aggressive (unique variance) cleanup.
    argument: ``-A``, position: 3
highpass: (a float, nipy default value: 100)
    cleanup motion confounds
    argument: ``-m -h %f``, 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
confound_file_1: (a file name)
    Include additional confound file.
    argument: ``-x %s``, position: 5
```

Outputs:

```
cleaned_functional_file: (an existing file name)
    Cleaned session data
```

63.4.3 FeatureExtractor

[Link to code](#)

Extract features (for later training and/or classifying)

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

Additional parameters to the command
argument: ``%s``
mel_ica: (an existing directory name)
Melodic output directory or directories
argument: ``%s``, position: -1

```

Outputs:

```

mel_ica: (an existing directory name)
Melodic output directory or directories
argument: ``%s``, position: -1

```

63.4.4 Training[Link to code](#)

Train the classifier based on your own FEAT/MELODIC output directory.

Inputs:

```

[Optional]
loo: (a boolean)
    full leave-one-out test with classifier training
    argument: ``-l``, position: 2
trained_wts_filestem: (a unicode string)
    trained-weights filestem, used for trained_wts_file and output
    directories
    argument: ``%s``, position: 1
mel_icas: (a list of items which are an existing directory name)
    Melodic output directories
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

trained_wts_file: (an existing file name)
    Trained-weights file

```

63.4.5 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:

```

[Optional]
mel_icas_in: (a list of items which are an existing directory name)
    Melodic output directories
    argument: ``%s``, position: -1

```

Outputs:

```
mel_icas_out: (a list of items which are an existing directory name)
    Hand labels for noise vs signal
    argument: ``%s``, position: -1
```

63.5 interfaces.fsl.maths

63.5.1 AR1Image

[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: 2

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
out_file: (a file name)
    image to write
    argument: ``%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)
    argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
    dimension to find AR(1) coefficient across
    argument: ``-%sar1``, position: 4
```

Outputs:

```
out_file: (an existing file name)
    image written after calculations
```

References:

None

63.5.2 ApplyMask

[Link to code](#)

Wraps the executable command `fslmaths`.

Use `fslmaths` to apply a binary mask to another image.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        image to operate on
        argument: ``%s``, position: 2
mask_file: (an existing file name)
        binary image defining mask space
        argument: ``-mas %s``, position: 4

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
             'NIFTI_GZ')
            FSL output type
out_file: (a file name)
        image to write
        argument: ``%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)
            argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
            change NaNs to zeros before doing anything
            argument: ``-nan``, position: 3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
        Environment variables
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
            datatype to use for calculations (default is float)
            argument: ``-dt %s``, position: 1
```

Outputs:

```
out_file: (an existing file name)
        image written after calculations
```

References:

None

63.5.3 BinaryMaths

[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: 2
operand_file: (an existing file name)
    second image to perform operation with
    argument: ``%s``, position: 5
    mutually_exclusive: operand_value
operation: ('add' or 'sub' or 'mul' or 'div' or 'rem' or 'max' or
    'min')
    operation to perform
    argument: ``-%s``, position: 4
operand_value: (a float)
    value to perform operation with
    argument: ``%.8f``, position: 5
    mutually_exclusive: operand_file

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
out_file: (a file name)
    image to write
    argument: ``%s``, position: -2
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, 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
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    argument: ``-odt %s``, position: -1
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    argument: ``-dt %s``, position: 1

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:

None

63.5.4 ChangeDataType

[Link to code](#)

Wraps the executable command `fslmaths`.

Use `fslmaths` to change the datatype of an image.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        argument: ``%s``, position: 2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                  or 'input')
        output data type
        argument: ``-odt %s``, position: -1

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
out_file: (a file name)
        image to write
        argument: ``%s``, position: -2
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        argument: ``-nan``, position: 3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
        Environment variables
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
        datatype to use for calculations (default is float)
        argument: ``-dt %s``, position: 1

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:

None

63.5.5 DilatelImage

[Link to code](#)

Wraps the executable command `fslmaths`.

Use `fslmaths` to perform a spatial dilation of an image.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        argument: ``%s``, position: 2
operation: ('mean' or 'modal' or 'max')
        filtering operation to perfoem in dilation
        argument: ``-dil%s``, position: 6

[Optional]
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

        Additional parameters to the command
        argument: ``%s``
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        argument: ``-nan``, 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
kernel_file: (an existing file name)
        use external file for kernel
        argument: ``%s``, position: 5
        mutually_exclusive: kernel_size
out_file: (a file name)
        image to write
        argument: ``%s``, position: -2
kernel_shape: ('3D' or '2D' or 'box' or 'boxv' or 'gauss' or 'sphere'
        or 'file')
        kernel shape to use
        argument: ``-kernel %s``, position: 4
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for output (default uses input type)
        argument: ``-odt %s``, position: -1
kernel_size: (a float)
        kernel size - voxels for box/boxv, mm for sphere, mm sigma for gauss
        argument: ``%.4f``, position: 5
        mutually_exclusive: kernel_file
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_GZ')
        FSL output type
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')
        datatype to use for calculations (default is float)
        argument: ``-dt %s``, position: 1

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:

None

63.5.6 ErodeImage[Link to code](#)Wraps the executable command `fslmaths`.Use `fslmaths` to perform a spatial erosion of an image.**Inputs:**

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        argument: ``%s``, position: 2

```

(continues on next page)

(continued from previous page)

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, 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', nipype default value: {})
    Environment variables
kernel_file: (an existing file name)
    use external file for kernel
    argument: ``%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
    argument: ``-kernel %s``, position: 4
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    argument: ``-odt %s``, position: -1
minimum_filter: (a boolean, nipype default value: False)
    if true, minimum filter rather than erosion by zeroing-out
    argument: ``%s``, position: 6
kernel_size: (a float)
    kernel size - voxels for box/boxv, mm for sphere, mm sigma for gauss
    argument: ``%.4f``, position: 5
    mutually_exclusive: kernel_file
out_file: (a file name)
    image to write
    argument: ``%s``, position: -2
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    argument: ``-dt %s``, position: 1

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:

None

63.5.7 IsotropicSmooth[Link to code](#)Wraps the executable command `fslmaths`.Use `fslmaths` to spatially smooth an image with a gaussian kernel.

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        argument: ``%s``, position: 2
sigma: (a float)
        sigma of smoothing kernel [mm]
        argument: ``-s %.5f``, position: 4
        mutually_exclusive: fwhm
fwhm: (a float)
        fwhm of smoothing kernel [mm]
        argument: ``-s %.5f``, position: 4
        mutually_exclusive: sigma

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
        datatype to use for calculations (default is float)
        argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
out_file: (a file name)
        image to write
        argument: ``%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)
        argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        argument: ``-nan``, position: 3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a 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

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:

None

63.5.8 MathsCommand[Link to code](#)Wraps the executable command `fslmaths`.**Inputs:**

```

[Mandatory]
in_file: (an existing file name)
        image to operate on

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``, position: 2

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
out_file: (a file name)
          image to write
          argument: ``%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)
              argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
            change NaNs to zeros before doing anything
            argument: ``-nan``, position: 3
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
        Environment variables
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
                 datatype to use for calculations (default is float)
                 argument: ``-dt %s``, position: 1

```

Outputs:

```

out_file: (an existing file name)
          image written after calculations

```

References:

None

63.5.9 MaxImage[Link to code](#)Wraps the executable 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)

```

(continues on next page)

(continued from previous page)

```

        image to operate on
        argument: ``%s``, position: 2

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
        datatype to use for calculations (default is float)
        argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
out_file: (a file name)
        image to write
        argument: ``%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)
        argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        argument: ``-nan``, position: 3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
        Environment variables
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
        dimension to max across
        argument: ``-%smax``, position: 4

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:

None

63.5.10 MaxnImage[Link to code](#)Wraps the executable 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
        argument: ``%s``, position: 2

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')

```

(continues on next page)

(continued from previous page)

```

        datatype to use for calculations (default is float)
        argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
             'NIFTI_GZ')
        FSL output type
out_file: (a file name)
        image to write
        argument: ``%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)
        argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        argument: ``-nan``, position: 3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
        Environment variables
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
        dimension to index max across
        argument: ``-%smxn``, position: 4

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:

None

63.5.11 MeanImage[Link to code](#)Wraps the executable 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
        argument: ``%s``, position: 2

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
        datatype to use for calculations (default is float)
        argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
             'NIFTI_GZ')
        FSL output type
out_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

        image to write
        argument: ``%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)
        argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        argument: ``-nan``, position: 3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
        Environment variables
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
        dimension to mean across
        argument: ``-%smean``, position: 4

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:

None

63.5.12 MedianImage[Link to code](#)Wraps the executable 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
        argument: ``%s``, position: 2

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
        datatype to use for calculations (default is float)
        argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
out_file: (a file name)
        image to write
        argument: ``%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)
        argument: ``-odt %s``, position: -1

```

(continues on next page)

(continued from previous page)

```

nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
    dimension to median across
    argument: ``-%smedian``, position: 4

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:

None

63.5.13 MinImage[Link to code](#)Wraps the executable 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
    argument: ``%s``, position: 2

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
out_file: (a file name)
    image to write
    argument: ``%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)
    argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
          dimension to min across
          argument: ``-%smin``, position: 4

```

Outputs:

```

out_file: (an existing file name)
          image written after calculations

```

References:

None

63.5.14 MultiImageMaths[Link to code](#)

Wraps the executable 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
          argument: ``%s``, position: 2
operand_files: (a list of items which are an existing file name)
               list of file names to plug into op string
op_string: (a string)
           python formatted string of operations to perform
           argument: ``%s``, position: 4

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
                  datatype to use for calculations (default is float)
                  argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
              FSL output type
out_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

        image to write
        argument: ``%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)
        argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        argument: ``-nan``, position: 3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

out_file: (an existing file name)
          image written after calculations

```

References:

None

63.5.15 PercentileImage[Link to code](#)Wraps the executable 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
        argument: ``%s``, position: 2

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
        datatype to use for calculations (default is float)
        argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')

```

(continues on next page)

(continued from previous page)

```

    FSL output type
perc: (0 <= a long integer <= 100)
    nth percentile (0-100) of FULL RANGE across dimension
    argument: ``%f``, position: 5
out_file: (a file name)
    image to write
    argument: ``%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)
    argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
    dimension to percentile across
    argument: ``-%sperc``, position: 4

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:

None

63.5.16 SpatialFilter[Link to code](#)Wraps the executable command `fslmaths`.Use `fslmaths` to spatially filter an image.**Inputs:**

```

[Mandatory]
in_file: (an existing file name)
    image to operate on
    argument: ``%s``, position: 2
operation: ('mean' or 'median' or 'meanu')
    operation to filter with
    argument: ``-f%s``, position: 6

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, position: 3

```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
kernel_file: (an existing file name)
             use external file for kernel
             argument: ``%s``, position: 5
             mutually_exclusive: kernel_size
out_file: (a file name)
          image to write
          argument: ``%s``, position: -2
kernel_shape: ('3D' or '2D' or 'box' or 'boxv' or 'gauss' or 'sphere'
              or 'file')
             kernel shape to use
             argument: ``-kernel %s``, position: 4
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                 or 'input')
             datatype to use for output (default uses input type)
             argument: ``-odt %s``, position: -1
kernel_size: (a float)
             kernel size - voxels for box/boxv, mm for sphere, mm sigma for gauss
             argument: ``%.4f``, position: 5
             mutually_exclusive: kernel_file
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
             FSL output type
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
                 datatype to use for calculations (default is float)
                 argument: ``-dt %s``, position: 1

```

Outputs:

```

out_file: (an existing file name)
          image written after calculations

```

References:

None

63.5.17 StdImage[Link to code](#)

Wraps the executable 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
         argument: ``%s``, position: 2

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                   or 'input')
                 datatype to use for calculations (default is float)

```

(continues on next page)

(continued from previous page)

```

        argument: ``-dt %s``, position: 1
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
out_file: (a file name)
        image to write
        argument: ``%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)
        argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
        change NaNs to zeros before doing anything
        argument: ``-nan``, position: 3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
        Environment variables
dimension: ('T' or 'X' or 'Y' or 'Z', nipy default value: T)
        dimension to standard deviate across
        argument: ``-%sstd``, position: 4

```

Outputs:

```

out_file: (an existing file name)
        image written after calculations

```

References:

None

63.5.18 TemporalFilter[Link to code](#)Wraps the executable 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
        argument: ``%s``, position: 2

[Optional]
lowpass_sigma: (a float, nipy default value: -1)
        lowpass filter sigma (in volumes)
        argument: ``%.6f``, position: 5
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
highpass_sigma: (a float, nipy default value: -1)
        highpass filter sigma (in volumes)
        argument: ``-bptf %.6f``, position: 4

```

(continues on next page)

(continued from previous page)

```

out_file: (a file name)
    image to write
    argument: ``%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)
    argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    argument: ``-dt %s``, position: 1

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:

None

63.5.19 Threshold[Link to code](#)Wraps the executable command `fslmaths`.Use `fslmaths` to apply a threshold to an image in a variety of ways.**Inputs:**

```

[Mandatory]
thresh: (a float)
    threshold value
    argument: ``%s``, position: 4
in_file: (an existing file name)
    image to operate on
    argument: ``%s``, position: 2

[Optional]
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for calculations (default is float)
    argument: ``-dt %s``, position: 1
out_file: (a file name)
    image to write
    argument: ``%s``, position: -2
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')

```

(continues on next page)

(continued from previous page)

```

        FSL output type
use_nonzero_voxels: (a boolean)
    use nonzero voxels to calculate robust range
    requires: use_robust_range
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, 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
use_robust_range: (a boolean)
    interpret thresh as percentage (0-100) of robust range
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
    or 'input')
    datatype to use for output (default uses input type)
    argument: ``-odt %s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
direction: ('below' or 'above', nipy default value: below)
    zero-out either below or above thresh value

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:

None

63.5.20 UnaryMaths[Link to code](#)Wraps the executable 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
    argument: ``%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
    argument: ``-%s``, position: 4

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
out_file: (a file name)
    image to write

```

(continues on next page)

(continued from previous page)

```

        argument: ``%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)
    argument: ``-odt %s``, position: -1
nan2zeros: (a boolean)
    change NaNs to zeros before doing anything
    argument: ``-nan``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a 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
internal_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
                    or 'input')
    datatype to use for calculations (default is float)
    argument: ``-dt %s``, position: 1

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

References:

None

63.6 interfaces.fsl.model

63.6.1 Cluster

[Link to code](#)Wraps the executable 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
    argument: ``--in=%s``
threshold: (a float)
    threshold for input volume

```

(continues on next page)

(continued from previous page)

```

        argument: ``--thresh=%.10f``

[Optional]
find_min: (a boolean, nipy default value: False)
    find minima instead of maxima
    argument: ``--min``
use_mm: (a boolean, nipy default value: False)
    use mm, not voxel, coordinates
    argument: ``--mm``
fractional: (a boolean, nipy default value: False)
    interprets the threshold as a fraction of the robust range
    argument: ``--fractional``
out_pval_file: (a boolean or a file name)
    filename for image output of log pvals
    argument: ``--opvals=%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
peak_distance: (a float)
    minimum distance between local maxima/minima, in mm (default 0)
    argument: ``--peakdist=%.10f``
cope_file: (a file name)
    cope volume
    argument: ``--cope=%s``
no_table: (a boolean, nipy default value: False)
    suppresses printing of the table info
    argument: ``--no_table``
num_maxima: (an integer (int or long))
    no of local maxima to report
    argument: ``--num=%d``
connectivity: (an integer (int or long))
    the connectivity of voxels (default 26)
    argument: ``--connectivity=%d``
out_index_file: (a boolean or a file name)
    output of cluster index (in size order)
    argument: ``--oindex=%s``
out_max_file: (a boolean or a file name)
    filename for output of max image
    argument: ``--omax=%s``
std_space_file: (a file name)
    filename for standard-space volume
    argument: ``--stdvol=%s``
volume: (an integer (int or long))
    number of voxels in the mask
    argument: ``--volume=%d``
out_localmax_txt_file: (a boolean or a file name)
    local maxima text file
    argument: ``--olmax=%s``
out_size_file: (a boolean or a file name)
    filename for output of size image
    argument: ``--osize=%s``
dlh: (a float)
    smoothness estimate = sqrt(det(Lambda))
    argument: ``--dlh=%.10f``
xfm_file: (a file name)
    filename for Linear: input->standard-space transform. Non-linear:

```

(continues on next page)

(continued from previous page)

```

    input->highres transform
    argument: ``--xfm=%s``
minclustersize: (a boolean, nipy default value: False)
    prints out minimum significant cluster size
    argument: ``--minclustersize``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
out_mean_file: (a boolean or a file name)
    filename for output of mean image
    argument: ``--omean=%s``
warpfield_file: (a file name)
    file containing warpfield
    argument: ``--warpvol=%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
out_localmax_vol_file: (a boolean or a file name)
    output of local maxima volume
    argument: ``--olmaxim=%s``
out_threshold_file: (a boolean or a file name)
    thresholded image
    argument: ``--othresh=%s``
pthreshold: (a float)
    p-threshold for clusters
    argument: ``--pthresh=%.10f``
    requires: dlh, volume

```

Outputs:

```

mean_file: (a file name)
    filename for output of mean image
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
threshold_file: (a file name)
    thresholded image
size_file: (a file name)
    filename for output of size image
pval_file: (a file name)
    filename for image output of log pvals
max_file: (a file name)
    filename for output of max image

```

References:

None

63.6.2 ContrastMgr[Link to code](#)Wraps the executable 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]
sigmasquareds: (an existing file name)
    summary of residuals, See Woolrich, et. al., 2001
param_estimates: (a list of items which are an existing file name)
    Parameter estimates for each column of the design matrix
corrections: (an existing file name)
    statistical corrections used within FILM modelling
dof_file: (an existing file name)
    degrees of freedom
tcon_file: (an existing file name)
    contrast file containing T-contrasts
    argument: ``%s``, position: -1

[Optional]
suffix: (a unicode string)
    suffix to put on the end of the cope filename before the contrast
    number, default is nothing
    argument: ``-suffix %s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
fcon_file: (an existing file name)
    contrast file containing F-contrasts
    argument: ``-f %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
contrast_num: (a long integer >= 1)
    contrast number to start labeling copes from
    argument: ``-cope``
```

Outputs:

```
fstats: (a list of items which are an existing file name)
    f-stat file for each contrast
copes: (a list of items which are an existing file name)
    Contrast estimates for each contrast
varcopes: (a list of items which are an existing file name)
    Variance estimates for each contrast
neffs: (a list of items which are an existing file name)
    neff file ?? for each contrast
zstats: (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
tstats: (a list of items which are an existing file name)
    t-stat file for each contrast
```

References:

None

63.6.3 DualRegression

[Link to code](#)

Wraps the executable command `dual_regression`.

Wrapper Script for Dual Regression Workflow

Examples

```
>>> dual_regression = DualRegression()
>>> dual_regression.inputs.in_files = ["functional.nii", "functional2.nii",
↳ "functional3.nii"]
>>> dual_regression.inputs.group_IC_maps_4D = "allFA.nii"
>>> dual_regression.inputs.des_norm = False
>>> dual_regression.inputs.one_sample_group_mean = True
>>> dual_regression.inputs.n_perm = 10
>>> dual_regression.inputs.out_dir = "my_output_directory"
>>> dual_regression.cmdline
'dual_regression allFA.nii 0 -1 10 my_output_directory functional.nii functional2.
↳ nii functional3.nii'
>>> dual_regression.run()
```

Inputs:

```
[Mandatory]
n_perm: (an integer (int or long))
    Number of permutations for randomise; set to 1 for just raw tstat
    output, set to 0 to not run randomise at all.
    argument: ``%i``, position: 5
in_files: (a list of items which are an existing file name)
    List all subjects' preprocessed, standard-space 4D datasets
    argument: ``%s``, position: -1
group_IC_maps_4D: (an existing file name)
    4D image containing spatial IC maps (melodic_IC) from the whole-
    group ICA analysis
    argument: ``%s``, position: 1

[Optional]
out_dir: (a directory name, nipy default value: output)
    This directory will be created to hold all output and logfiles
    argument: ``%s``, position: 6
design_file: (an existing file name)
    Design matrix for final cross-subject modelling with randomise
    argument: ``%s``, position: 3
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
des_norm: (a boolean, nipy default value: True)
    Whether to variance-normalise the timecourses used as the stage-2
    regressors; True is default and recommended
    argument: ``%i``, position: 2
con_file: (an existing file name)
    Design contrasts for final cross-subject modelling with randomise
    argument: ``%s``, position: 4
one_sample_group_mean: (a boolean)
    perform 1-sample group-mean test instead of generic permutation test
```

(continues on next page)

(continued from previous page)

```
argument: ``-1``, 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
```

Outputs:

```
out_dir: (an existing directory name)
```

References:

None

63.6.4 FEAT

[Link to code](#)Wraps the executable 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
         argument: ``%s``, position: 0

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
```

Outputs:

```
feat_dir: (an existing directory name)
```

References:

None

63.6.5 FEATModel

[Link to code](#)Wraps the executable command `feat_model`.Uses FSL `feat_model` to generate `design.mat` files**Inputs:**

```
[Mandatory]
fsf_file: (an existing file name)
         File specifying the feat design spec file
```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``, position: 0
ev_files: (a list of items which are an existing file name)
        Event spec files generated by level1design
        argument: ``%s``, position: 1

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
        Environment variables
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``

```

Outputs:

```

design_file: (an existing file name)
        Mat file containing ascii matrix for design
con_file: (an existing file name)
        Contrast file containing contrast vectors
design_cov: (an existing file name)
        Graphical representation of design covariance
design_image: (an existing file name)
        Graphical representation of design matrix
fcon_file: (a file name)
        Contrast file containing contrast vectors

```

References:

None

63.6.6 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]
reg_dof: (an integer (int or long), nipy default value: 12)
        registration degrees of freedom

```

Outputs:

```

fsf_file: (an existing file name)
        FSL feat specification file

```

63.6.7 FILMGLS

[Link to code](#)

Wraps the executable 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
         argument: ``%s``, position: -3

[Optional]
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
smooth_autocorr: (a boolean)
                 Smooth auto corr estimates
                 argument: ``-sa``
full_data: (a boolean)
           output full data
           argument: ``-v``
output_pwdata: (a boolean)
               output prewhitened data and average design matrix
               argument: ``-output_pwdata``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
              FSL output type
use_pava: (a boolean)
          estimates autocorr using PAVA
          argument: ``-pava``
tukey_window: (an integer (int or long))
              tukey window size to estimate autocorr
              argument: ``-tukey %d``
mutually_exclusive: autocorr_estimate_only, fit_armodel,
                    tukey_window, multitaper_product, use_pava, autocorr_noestimate
```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a 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
design_file: (an existing file name)
        design matrix file
        argument: ``%s``, position: -2
results_dir: (a directory name, nipyne default value: results)
        directory to store results in
        argument: ``-rn %s``
mask_size: (an integer (int or long))
        susan mask size
        argument: ``-ms %d``
autocorr_estimate_only: (a boolean)
        perform autocorrelation estimation only
        argument: ``-ac``
        mutually_exclusive: autocorr_estimate_only, fit_armodel,
        tukey_window, multitaper_product, use_pava, autocorr_noestimate
threshold: (a floating point number >= 0.0, nipyne default value:
        1000.0)
        threshold
        argument: ``%f``, position: -1
autocorr_noestimate: (a boolean)
        do not estimate autocorrs
        argument: ``-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
        argument: ``-epith %d``
fit_armodel: (a boolean)
        fits autoregressive model - default is to use tukey with
        M=sqrt(numvols)
        argument: ``-ar``
        mutually_exclusive: autocorr_estimate_only, fit_armodel,
        tukey_window, multitaper_product, use_pava, autocorr_noestimate
multitaper_product: (an integer (int or long))
        multitapering with slepian tapers and num is the time-bandwidth
        product
        argument: ``-mt %d``
        mutually_exclusive: autocorr_estimate_only, fit_armodel,
        tukey_window, multitaper_product, use_pava, autocorr_noestimate

```

Outputs:

```

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
corrections: (an existing file name)
        statistical corrections used within FILM modeling
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

```

(continues on next page)

(continued from previous page)

```
logfile: (an existing file name)
        FILM run logfile
thresholdac: (an existing file name)
             The FILM autocorrelation parameters
```

References:

None

63.6.8 FLAMEO[Link to code](#)

Wraps the executable 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]
design_file: (an existing file name)
            design matrix file
            argument: ``--designfile=%s``
cope_file: (an existing file name)
            cope regressor data file
            argument: ``--copefile=%s``
run_mode: ('fe' or 'ols' or 'flame1' or 'flame12')
            inference to perform
            argument: ``--runmode=%s``
mask_file: (an existing file name)
            mask file
            argument: ``--maskfile=%s``
t_con_file: (an existing file name)
            ascii matrix specifying t-contrasts
            argument: ``--tcontrastsfile=%s``
cov_split_file: (an existing file name)
            ascii matrix specifying the groups the covariance is split into
            argument: ``--covsplitfile=%s``

[Optional]
```

(continues on next page)

(continued from previous page)

```

n_jumps: (an integer (int or long))
    number of jumps made by mcmc
    argument: ``--n_jumps=%d``
infer_outliers: (a boolean)
    infer outliers - not for fe
    argument: ``--inferoutliers``
outlier_iter: (an integer (int or long))
    Number of max iterations to use when inferring outliers. Default is
    12.
    argument: ``--ioni=%d``
var_cope_file: (an existing file name)
    varcope weightings data file
    argument: ``--varcopefile=%s``
f_con_file: (an existing file name)
    ascii matrix specifying f-contrasts
    argument: ``--fcontrastsfile=%s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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
    argument: ``--sigma_dofs=%d``
fix_mean: (a boolean)
    fix mean for tfit
    argument: ``--fixmean``
environ: (a dictionary with keys which are a 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
dof_var_cope_file: (an existing file name)
    dof data file for varcope data
    argument: ``--dofvarcopefile=%s``
burnin: (an integer (int or long))
    number of jumps at start of mcmc to be discarded
    argument: ``--burnin=%d``
no_pe_outputs: (a boolean)
    do not output pe files
    argument: ``--nopeoutput``
sample_every: (an integer (int or long))
    number of jumps for each sample
    argument: ``--sampleevery=%d``
log_dir: (a directory name, nipype default value: stats)
    argument: ``--ld=%s``

```

Outputs:

```

fstats: (a list of items which are an existing file name)
    f-stat file for each contrast
copes: (a list of items which are an existing file name)
    Contrast estimates for each contrast
res4d: (a list of items which are an existing file name)
    Model fit residual mean-squared error for each time point
tdof: (a list of items which are an existing file name)

```

(continues on next page)

(continued from previous page)

```

    temporal dof file for each contrast
zfstats: (a list of items which are an existing file name)
    z stat file for each f contrast
pes: (a list of items which are an existing file name)
    Parameter estimates for each column of the design matrix for each
    voxel
mrefvars: (a list of items which are an existing file name)
    mean random effect variances 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
stats_dir: (a directory name)
    directory storing model estimation output
zstats: (a list of items which are an existing file name)
    z-stat file for each contrast
tstats: (a list of items which are an existing file name)
    t-stat file for each contrast

```

References:

None None

63.6.9 GLM[Link to code](#)Wraps the executable 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)
    argument: ``-d %s``, position: 2
in_file: (an existing file name)
    input file name (text matrix or 3D/4D image file)
    argument: ``-i %s``, position: 1

[Optional]
out_vnscales_name: (a file name)
    output file name for scaling factors for variance normalisation
    argument: ``--out_vnscales=%s``
mask: (an existing file name)
    mask image file name if input is image
    argument: ``-m %s``

```

(continues on next page)

(continued from previous page)

```

out_varcb_name: (a file name)
    output file name for variance of COPEs
    argument: ``--out_varcb=%s``
des_norm: (a boolean)
    switch on normalization of the design matrix columns to unit std
    deviation
    argument: ``--des_norm``
out_z_name: (a file name)
    output file name for Z-stats (either as txt or image)
    argument: ``--out_z=%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
out_pf_name: (a file name)
    output file name for p-value for full model fit
    argument: ``--out_pf=%s``
out_cope: (a file name)
    output file name for COPE (either as txt or image)
    argument: ``--out_cope=%s``
out_res_name: (a file name)
    output file name for residuals
    argument: ``--out_res=%s``
out_p_name: (a file name)
    output file name for p-values of Z-stats (either as text file or
    image)
    argument: ``--out_p=%s``
out_data_name: (a file name)
    output file name for pre-processed data
    argument: ``--out_data=%s``
demean: (a boolean)
    switch on demeaning of design and data
    argument: ``--demean``
var_norm: (a boolean)
    perform MELODIC variance-normalisation on data
    argument: ``--vn``
out_sigsq_name: (a file name)
    output file name for residual noise variance sigma-square
    argument: ``--out_sigsq=%s``
out_file: (a file name)
    filename for GLM parameter estimates (GLM betas)
    argument: ``-o %s``, position: 3
dof: (an integer (int or long))
    set degrees of freedom explicitly
    argument: ``--dof=%d``
dat_norm: (a boolean)
    switch on normalization of the data time series to unit std
    deviation
    argument: ``--dat_norm``
contrasts: (an existing file name)
    matrix of t-statics contrasts
    argument: ``-c %s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
out_f_name: (a file name)
    output file name for F-value of full model fit

```

(continues on next page)

(continued from previous page)

```

        argument: ``--out_f=%s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
out_t_name: (a file name)
        output file name for t-stats (either as txt or image
        argument: ``--out_t=%s``

```

Outputs:

```

out_f: (a list of items which are an existing file name)
        output file name for F-value of full model fit
out_z: (a list of items which are an existing file name)
        output file name for COPEs (either as text file or image)
out_file: (an existing file name)
        file name of GLM parameters (if generated)
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_data: (a list of items which are an existing file name)
        output file for preprocessed data
out_vnscales: (a list of items which are an existing file name)
        output file name for scaling factors for variance normalisation
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_cope: (a list of items which are an existing file name)
        output file name for COPEs (either as text file or image)
out_sigsq: (a list of items which are an existing file name)
        output file name for residual noise variance sigma-square
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

```

References:

None

63.6.10 L2Model[Link to code](#)

Generate subject specific second level model

Examples

```

>>> from nipyne.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

```

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

```

63.6.11 Level1Design

[Link to code](#)

Generate FEAT specific files

Examples

```

>>> levelldesign = Level1Design()
>>> levelldesign.inputs.interscan_interval = 2.5
>>> levelldesign.inputs.bases = {'dgamma':{'derivs': False}}
>>> levelldesign.inputs.session_info = 'session_info.npz'
>>> levelldesign.run()

```

Inputs:

```

[Mandatory]
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 FIIMGLS by setting autocorr_noestimate to
    True
session_info: (any value)
    Session specific information generated by ``modelgen.SpecifyModel``
interscan_interval: (a float)
    Interscan interval (in secs)
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 'gammasigma' or 'gammadelay' 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}}

[Optional]
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), nipy default value: {})
    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.

```

(continues on next page)

(continued from previous page)

```

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.

```

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

```

63.6.12 MELODIC

[Link to code](#)Wraps the executable 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)
    argument: ``-i %s``, position: 0

[Optional]
out_orig: (a boolean)
    output the original ICs
    argument: ``--Oorig``
max_restart: (an integer (int or long))
    maximum number of restarts
    argument: ``--maxrestart=%d``
t_des: (an existing file name)
    design matrix across time-domain
    argument: ``--Tdes=%s``
bg_threshold: (a float)
    brain/non-brain threshold used to mask non-brain voxels, as a
    percentage (only if --nobet selected)
    argument: ``--bgthreshold=%f``
bg_image: (an existing file name)
    specify background image for report (default: mean image)
    argument: ``--bgimage=%s``
no_mask: (a boolean)
    switch off masking
    argument: ``--nomask``
out_dir: (a directory name)
    output directory name
    argument: ``-o %s``
out_unmix: (a boolean)
    output unmixing matrix
    argument: ``--Ounmix``
out_white: (a boolean)
    output whitening/dewhitening matrices
    argument: ``--Owhite``
var_norm: (a boolean)
    switch off variance normalization
    argument: ``--vn``
epsilonS: (a float)
    minimum error change for rank-1 approximation in TICA
    argument: ``--epsS=%f``
rem_cmp: (a list of items which are an integer (int or long))
    component numbers to remove
    argument: ``-f %d``
no_mm: (a boolean)
    switch off mixture modelling on IC maps
    argument: ``--no_mm``
report_maps: (a unicode string)
    control string for spatial map images (see slicer)
    argument: ``--report_maps=%s``
dim: (an integer (int or long))
    dimensionality reduction into #num dimensions (default: automatic
    estimation)
    argument: ``-d %d``
out_pca: (a boolean)
    output PCA results
    argument: ``--Opca``
remove_deriv: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        removes every second entry in paradigm file (EV derivatives)
        argument: ``--remove_deriv``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
              FSL output type
t_con: (an existing file name)
        t-contrast matrix across time-domain
        argument: ``--Tcon=%s``
epsilon: (a float)
        minimum error change
        argument: ``--eps=%f``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
log_power: (a boolean)
        calculate log of power for frequency spectrum
        argument: ``--logPower``
out_mean: (a boolean)
        output mean volume
        argument: ``--Omean``
out_all: (a boolean)
        output everything
        argument: ``--Oall``
cov_weight: (a float)
        voxel-wise weights for the covariance matrix (e.g. segmentation
        information)
        argument: ``--covarweight=%f``
s_des: (an existing file name)
        design matrix across subject-domain
        argument: ``--Sdes=%s``
out_stats: (a boolean)
        output thresholded maps and probability maps
        argument: ``--Ostats``
mask: (an existing file name)
        file name of mask for thresholding
        argument: ``-m %s``
migp_factor: (an integer (int or long))
        Internal Factor of mem-threshold relative to number of Eigenmaps
        (default: 2)
        argument: ``--migp_factor %d``
migp: (a boolean)
        switch on MIGP data reduction
        argument: ``--migp``
migpN: (an integer (int or long))
        number of internal Eigenmaps
        argument: ``--migpN %d``
smode: (an existing file name)
        matrix of session modes for report generation
        argument: ``--smode=%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
dim_est: (a unicode string)
        use specific dim. estimation technique: lap, bic, mdl, aic, mean
        (default: lap)
        argument: ``--dimest=%s``

```

(continues on next page)

(continued from previous page)

```

mix: (an existing file name)
    mixing matrix for mixture modelling / filtering
    argument: ``--mix=%s``
mign_shuffle: (a boolean)
    randomise MIGN file order (default: TRUE)
    argument: ``--mign_shuffle``
num_ICs: (an integer (int or long))
    number of IC's to extract (for deflation approach)
    argument: ``-n %d``
report: (a boolean)
    generate Melodic web report
    argument: ``--report``
non_linearity: (a unicode string)
    nonlinearity: gauss, tanh, pow3, pow4
    argument: ``--nl=%s``
s_con: (an existing file name)
    t-contrast matrix across subject-domain
    argument: ``--Scon=%s``
ICs: (an existing file name)
    filename of the IC components file for mixture modelling
    argument: ``--ICs=%s``
maxit: (an integer (int or long))
    maximum number of iterations before restart
    argument: ``--maxit=%d``
mm_thresh: (a float)
    threshold for Mixture Model based inference
    argument: ``--mmthresh=%f``
no_bet: (a boolean)
    switch off BET
    argument: ``--nobet``
sep_whiten: (a boolean)
    switch on separate whitening
    argument: ``--sep_whiten``
tr_sec: (a float)
    TR in seconds
    argument: ``--tr=%f``
approach: (a unicode string)
    approach for decomposition, 2D: defl, symm (default), 3D: tica
    (default), concat
    argument: ``-a %s``
update_mask: (a boolean)
    switch off mask updating
    argument: ``--update_mask``
pbsc: (a boolean)
    switch off conversion to percent BOLD signal change
    argument: ``--pbsc``
sep_vn: (a boolean)
    switch off joined variance normalization
    argument: ``--sep_vn``

```

Outputs:

```

report_dir: (an existing directory name)
out_dir: (an existing directory name)

```

References:

None

63.6.13 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 nipy.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)
```

Outputs:

```
design_con: (an existing file name)
            design t-contrast file
design_grp: (an existing file name)
            design group file
design_mat: (an existing file name)
            design matrix file
design_fts: (an existing file name)
            design f-contrast file
```

63.6.14 Randomise

[Link to code](#)

Wraps the executable 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 "randomise" -d design.mat -t design.con -m mask.nii'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        4D input file
        argument: ``-i %s``, position: 0

[Optional]
fcon: (an existing file name)
      f contrasts file
      argument: ``-f %s``
tfce_E: (a float)
        TFCE extent parameter (default=0.5)
        argument: ``--tfce_E=%.2f``
vox_p_values: (a boolean)
              output voxelwise (corrected and uncorrected) p-value images
              argument: ``-x``
num_perm: (an integer (int or long))
          number of permutations (default 5000, set to 0 for exhaustive)
          argument: ``-n %d``
show_total_perms: (a boolean)
                  print out how many unique permutations would be generated and exit
                  argument: ``-q``
one_sample_group_mean: (a boolean)
                       perform 1-sample group-mean test instead of generic permutation test
                       argument: ``-l``
tfce_H: (a float)
        TFCE height parameter (default=2)
        argument: ``--tfce_H=%.2f``
tcon: (an existing file name)
      t contrasts file
      argument: ``-t %s``, 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
p_vec_n_dist_files: (a boolean)
                   output permutation vector and null distribution text files
                   argument: ``-P``
var_smooth: (an integer (int or long))
            use variance smoothing (std is in mm)
            argument: ``-v %d``
raw_stats_imgs: (a boolean)
                output raw ( unpermuted ) statistic images
                argument: ``-R``
tfce_C: (a float)
        TFCE connectivity (6 or 26; default=6)
        argument: ``--tfce_C=%.2f``
show_info_parallel_mode: (a boolean)
                         print out information required for parallel mode and exit
```

(continues on next page)

(continued from previous page)

```

        argument: ``-Q``
demean: (a boolean)
        demean data temporally before model fitting
        argument: ``-D``
mask: (an existing file name)
        mask image
        argument: ``-m %s``
cm_thresh: (a float)
        carry out cluster-mass-based thresholding
        argument: ``-C %.1f``
tfce2D: (a boolean)
        carry out Threshold-Free Cluster Enhancement with 2D optimisation
        argument: ``--T2``
x_block_labels: (an existing file name)
        exchangeability block labels file
        argument: ``-e %s``
f_cm_thresh: (a float)
        carry out f cluster-mass thresholding
        argument: ``-S %.2f``
seed: (an integer (int or long))
        specific integer seed for random number generator
        argument: ``--seed=%d``
f_only: (a boolean)
        calculate f-statistics only
        argument: ``--f_only``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
c_thresh: (a float)
        carry out cluster-based thresholding
        argument: ``-c %.1f``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
tfce: (a boolean)
        carry out Threshold-Free Cluster Enhancement
        argument: ``-T``
base_name: (a unicode string, nipy default value: randomise)
        the rootname that all generated files will have
        argument: ``-o "%s"`, position: 1
f_c_thresh: (a float)
        carry out f cluster thresholding
        argument: ``-F %.2f``
design_mat: (an existing file name)
        design matrix file
        argument: ``-d %s`, position: 2

```

Outputs:

```

t_corrected_p_files: (a list of items which are an existing file
                     name)
        t contrast FWE (Family-wise error) corrected p values files
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

```

(continues on next page)

(continued from previous page)

```

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
fstat_files: (a list of items which are an existing file name)
              f contrast raw statistic

```

References:

None

63.6.15 SMM[Link to code](#)Wraps the executable 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
      argument: ``--mask="%s"`, position: 1
spatial_data_file: (an existing file name)
                   statistics spatial map
                   argument: ``--sdf="%s"`, position: 0

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
             FSL output type
no_deactivation_class: (a boolean)
                       enforces no deactivation class
                       argument: ``--zfstatmode`, position: 2
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s`
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables

```

Outputs:

```

null_p_map: (an existing file name)
deactivation_p_map: (an existing file name)
activation_p_map: (an existing file name)

```

References:

None

63.6.16 SmoothEstimate[Link to code](#)

Wraps the executable 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
'smoolest --mask=mask.nii --zstat=zstat1.nii.gz'
```

Inputs:

```
[Mandatory]
mask_file: (an existing file name)
            brain mask volume
            argument: ``--mask=%s``
dof: (an integer (int or long))
      number of degrees of freedom
      argument: ``--dof=%d``
      mutually_exclusive: zstat_file

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
residual_fit_file: (an existing file name)
                   residual-fit image file
                   argument: ``--res=%s``
                   requires: dof
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
zstat_file: (an existing file name)
            zstat image file
            argument: ``--zstat=%s``
            mutually_exclusive: dof
environ: (a dictionary with keys which are a 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
```

Outputs:

```
volume: (an integer (int or long))
         number of voxels in mask
resels: (a float)
         number of resels
dlh: (a float)
      smoothness estimate sqrt(det(Lambda))
```

References:

None

63.6.17 load_template()

[Link to code](#)

Load a template from the model_templates directory

Parameters

name [str] The name of the file to load

Returns

template : string.Template

63.7 interfaces.fsl.possum

63.7.1 B0Calc

[Link to code](#)

Wraps the executable 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 nipyype.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 --chi0=4.000000e-
↪07 -d -9.450000e-06 --extendboundary=1.00 --b0x=0.00 --gx=0.0000 --b0y=0.00 --
↪gy=0.0000 --b0=3.00 --gz=0.0000'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         filename of input image (usually a tissue/air segmentation)
         argument: ``-i %s``, position: 0

[Optional]
out_file: (a file name)
         filename of B0 output volume
         argument: ``-o %s``, position: 1
chi_air: (a float, nipyype default value: 4e-07)
         susceptibility of air
         argument: ``--chi0=%e``
x_grad: (a float, nipyype default value: 0.0)
         Value for zeroth-order x-gradient field (per mm)
         argument: ``--gx=%0.4f``
compute_xyz: (a boolean, nipyype default value: False)
             calculate and save all 3 field components (i.e. x,y,z)
             argument: ``--xyz``
environ: (a dictionary with keys which are a 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: {})
```

(continues on next page)

(continued from previous page)

```

    Environment variables
y_grad: (a float, nipy default value: 0.0)
    Value for zeroth-order y-gradient field (per mm)
    argument: ``--gy=%0.4f``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
delta: (a float, nipy default value: -9.45e-06)
    Delta value (chi_tissue - chi_air)
    argument: ``-d %e``
z_grad: (a float, nipy default value: 0.0)
    Value for zeroth-order z-gradient field (per mm)
    argument: ``--gz=%0.4f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
y_b0: (a float, nipy default value: 0.0)
    Value for zeroth-order b0 field (y-component), in Tesla
    argument: ``--b0y=%0.2f``
    mutually_exclusive: xyz_b0
extendboundary: (a float, nipy default value: 1.0)
    Relative proportion to extend voxels at boundary
    argument: ``--extendboundary=%0.2f``
x_b0: (a float, nipy default value: 0.0)
    Value for zeroth-order b0 field (x-component), in Tesla
    argument: ``--b0x=%0.2f``
    mutually_exclusive: xyz_b0
directconv: (a boolean, nipy default value: False)
    use direct (image space) convolution, not FFT
    argument: ``--directconv``
xyz_b0: (a tuple of the form: (a float, a float, a float))
    Zeroth-order B0 field in Tesla
    argument: ``--b0x=%0.2f --b0y=%0.2f --b0z=%0.2f``
    mutually_exclusive: x_b0, y_b0, z_b0
z_b0: (a float, nipy default value: 1.0)
    Value for zeroth-order b0 field (z-component), in Tesla
    argument: ``--b0z=%0.2f``
    mutually_exclusive: xyz_b0

```

Outputs:

```

out_file: (an existing file name)
    filename of B0 output volume

```

References:

None

63.8 interfaces.fsl.preprocess

63.8.1 ApplyWarp

[Link to code](#)

Wraps the executable command applywarp.

FSL's applywarp wrapper to apply the results of a FNIRT registration

Examples

```
>>> from nipy.interfaces import fsl
>>> from nipy.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]
ref_file: (an existing file name)
          reference image
          argument: ``--ref=%s``, position: 1
in_file: (an existing file name)
          image to be warped
          argument: ``--in=%s``, position: 0

[Optional]
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
out_file: (a file name)
          output filename
          argument: ``--out=%s``, position: 2
abswarp: (a boolean)
          treat warp field as absolute:  $x' = w(x)$ 
          argument: ``--abs``
          mutually_exclusive: relwarp
superlevel: ('a' or an integer (int or long))
            level of intermediary supersampling, a for 'automatic' or integer
            level. Default = 2
            argument: ``--superlevel=%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
interp: ('nn' or 'trilinear' or 'sinc' or 'spline')
        interpolation method
        argument: ``--interp=%s``, position: -2
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
datatype: ('char' or 'short' or 'int' or 'float' or 'double')
          Force output data type [char short int float double].
          argument: ``--datatype=%s``
mask_file: (an existing file name)
            filename for mask image (in reference space)
            argument: ``--mask=%s``
field_file: (an existing file name)
            file containing warp field
            argument: ``--warp=%s``
relwarp: (a boolean)
          treat warp field as relative:  $x' = x + w(x)$ 
          argument: ``--rel``, position: -1
          mutually_exclusive: abswarp
```

(continues on next page)

(continued from previous page)

```

premat: (an existing file name)
        filename for pre-transform (affine matrix)
        argument: ``--premat=%s``
postmat: (an existing file name)
        filename for post-transform (affine matrix)
        argument: ``--postmat=%s``
supersample: (a boolean)
        intermediary supersampling of output, default is off
        argument: ``--super``

```

Outputs:

```

out_file: (an existing file name)
        Warped output file

```

References:

None

63.8.2 ApplyXFM[Link to code](#)Wraps the executable 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]
reference: (an existing file name)
        reference file
        argument: ``-ref %s``, position: 1
in_file: (an existing file name)
        input file
        argument: ``-in %s``, position: 0

[Optional]
padding_size: (an integer (int or long))
        for applyxfm: interpolates outside image by size
        argument: ``-paddingsize %d``
wmcoords: (a file name)
        white matter boundary coordinates for BBR cost function
        argument: ``-wmcoords %s``
in_matrix_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

        input 4x4 affine matrix
        argument: ``-init %s``
in_weight: (an existing file name)
        File for input weighting volume
        argument: ``-inweight %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
        argument: ``-searchrz %s``
wm_seg: (a file name)
        white matter segmentation volume needed by BBR cost function
        argument: ``-wmseg %s``
min_sampling: (a float)
        set minimum voxel dimension for sampling
        argument: ``-minsampling %f``
pedir: (an integer (int or long))
        phase encode direction of EPI - 1/2/3=x/y/z & -1/-2/-3=-x/-y/-z
        argument: ``-pedir %d``
ref_weight: (an existing file name)
        File for reference weighting volume
        argument: ``-refweight %s``
interp: ('trilinear' or 'nearestneighbour' or 'sinc' or 'spline')
        final interpolation method used in reslicing
        argument: ``-interp %s``
uses_qform: (a boolean)
        initialize using sform or qform
        argument: ``-usesqform``
searchr_y: (a list of from 2 to 2 items which are an integer (int or
        long))
        search angles along y-axis, in degrees
        argument: ``-searchry %s``
cost_func: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
        'leastsq' or 'labeldiff' or 'bbr')
        cost function
        argument: ``-searchcost %s``
coarse_search: (an integer (int or long))
        coarse search delta angle
        argument: ``-coarsesearch %d``
echospacing: (a float)
        value of EPI echo spacing - units of seconds
        argument: ``-echospacing %f``
out_file: (a file name)
        registered output file
        argument: ``-out %s``, position: 2
no_search: (a boolean)
        set all angular searches to ranges 0 to 0
        argument: ``-nosearch``
bbrslope: (a float)
        value of bbr slope
        argument: ``-bbrslope %f``
out_matrix_file: (a file name)
        output affine matrix in 4x4 asciiii format
        argument: ``-omat %s``, position: 3
rigid2D: (a boolean)
        use 2D rigid body mode - ignores dof
        argument: ``-2D``
display_init: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        display initial matrix
        argument: ``-displayinit``
no_resample_blur: (a boolean)
        do not use blurring on downsampling
        argument: ``-noresampblur``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
datatype: ('char' or 'short' or 'int' or 'float' or 'double')
        force output data type
        argument: ``-datatype %s``
sinc_width: (an integer (int or long))
        full-width in voxels
        argument: ``-sincwidth %d``
bins: (an integer (int or long))
        number of histogram bins
        argument: ``-bins %d``
apply_isoxfm: (a float)
        as applyxfm but forces isotropic resampling
        argument: ``-applyisoxfm %f``
        mutually_exclusive: apply_xfm
bbrtype: ('signed' or 'global_abs' or 'local_abs')
        type of bbr cost function: signed [default], global_abs, local_abs
        argument: ``-bbrtype %s``
cost: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
       'leastsq' or 'labeldiff' or 'bbr')
        cost function
        argument: ``-cost %s``
no_clamp: (a boolean)
        do not use intensity clamping
        argument: ``-noclamp``
no_resample: (a boolean)
        do not change input sampling
        argument: ``-noresample``
fieldmap: (a file name)
        fieldmap image in rads/s - must be already registered to the
        reference image
        argument: ``-fieldmap %s``
schedule: (an existing file name)
        replaces default schedule
        argument: ``-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
sinc_window: ('rectangular' or 'hanning' or 'blackman')
        sinc window
        argument: ``-sincwindow %s``
fine_search: (an integer (int or long))
        fine search delta angle
        argument: ``-finesearch %d``
out_log: (a file name)
        output log
        requires: save_log
force_scaling: (a boolean)
        force rescaling even for low-res images
        argument: ``-forcescaling``

```

(continues on next page)

(continued from previous page)

```

searchr_x: (a list of from 2 to 2 items which are an integer (int or
            long))
            search angles along x-axis, in degrees
            argument: ``-searchrx %s``
dof: (an integer (int or long))
      number of transform degrees of freedom
      argument: ``-dof %d``
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
            argument: ``-applyxfm``
save_log: (a boolean)
           save to log file
fieldmapmask: (a file name)
              mask for fieldmap image
              argument: ``-fieldmapmask %s``
wmnorms: (a file name)
          white matter boundary normals for BBR cost function
          argument: ``-wmnorms %s``
angle_rep: ('quaternion' or 'euler')
            representation of rotation angles
            argument: ``-anglerep %s``
bgvalue: (a float)
          use specified background value for points outside FOV
          argument: ``-setbackground %f``
verbose: (an integer (int or long))
          verbose mode, 0 is least
          argument: ``-verbose %d``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``

```

Outputs:

```

out_matrix_file: (an existing file name)
                 path/name of calculated affine transform (if generated)
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)

```

References:

None

63.8.3 BET[Link to code](#)

Wraps the executable command bet.

FSL BET wrapper 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
        argument: ``%s``, position: 0

[Optional]
robust: (a boolean)
        robust brain centre estimation (iterates BET several times)
        argument: ``-R``
        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)
        argument: ``-S``
        mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
padding: (a boolean)
        improve BET if FOV is very small in Z (by temporarily padding end
        slices)
        argument: ``-Z``
        mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
t2_guided: (a file name)
        as with creating surfaces, when also feeding in non-brain-extracted
        T2 (includes registrations)
        argument: ``-A2 %s``
        mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
vertical_gradient: (a float)
        vertical gradient in fractional intensity threshold (-1, 1)
        argument: ``-g %.2f``
environ: (a dictionary with keys which are a bytes or None or a 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
        argument: ``-f %.2f``
center: (a list of at most 3 items which are an integer (int or
        long))
        center of gravity in voxels
        argument: ``-c %s``
functional: (a boolean)
        apply to 4D fMRI data

```

(continues on next page)

(continued from previous page)

```

        argument: ``-F``
        mutually_exclusive: functional, reduce_bias, robust, padding,
            remove_eyes, surfaces, t2_guided
threshold: (a boolean)
    apply thresholding to segmented brain image and mask
    argument: ``-t``
mask: (a boolean)
    create binary mask image
    argument: ``-m``
out_file: (a file name)
    name of output skull stripped image
    argument: ``%s``, position: 1
radius: (an integer (int or long))
    head radius
    argument: ``-r %d``
skull: (a boolean)
    create skull image
    argument: ``-s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
surfaces: (a boolean)
    run bet2 and then betsurf to get additional skull and scalp surfaces
    (includes registrations)
    argument: ``-A``
    mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
mesh: (a boolean)
    generate a vtk mesh brain surface
    argument: ``-e``
reduce_bias: (a boolean)
    bias field and neck cleanup
    argument: ``-B``
    mutually_exclusive: functional, reduce_bias, robust, padding,
        remove_eyes, surfaces, t2_guided
no_output: (a boolean)
    Don't generate segmented output
    argument: ``-n``
outline: (a boolean)
    create surface outline image
    argument: ``-o``

```

Outputs:

```

outline_file: (a file name)
    path/name of outline file (if generated)
skull_mask_file: (a file name)
    path/name of skull mask (if generated)
outskull_mesh_file: (a file name)
    path/name of outskull mesh outline (if generated)
meshfile: (a file name)
    path/name of vtk mesh file (if generated)
outskull_mask_file: (a file name)
    path/name of outskull mask (if generated)
out_file: (a file name)
    path/name of skullstripped file (if generated)
inskull_mesh_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

    path/name of inskull mesh outline (if generated)
    outskin_mesh_file: (a file name)
    path/name of outskin mesh outline (if generated)
    mask_file: (a file name)
    path/name of binary brain mask (if generated)
    inskull_mask_file: (a file name)
    path/name of inskull mask (if generated)
    outskin_mask_file: (a file name)
    path/name of outskin mask (if generated)

```

References:

None

63.8.4 FAST[Link to code](#)

Wraps the executable command fast.

FSL FAST wrapper for segmentation 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
    argument: ``%s``, position: -1

[Optional]
other_priors: (a list of from 3 to 3 items which are a file name)
    alternative prior images
    argument: ``-A %s``
segment_iters: (1 <= a long integer <= 50)
    number of segmentation-initialisation iterations
    argument: ``-W %d``
init_transform: (an existing file name)
    <standard2input.mat> initialise using priors
    argument: ``-a %s``
iters_afterbias: (1 <= a long integer <= 20)
    number of main-loop iterations after bias-field removal
    argument: ``-O %d``
out_basename: (a file name)
    base name of output files
    argument: ``-o %s``
output_biascorrected: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        output restored image (bias-corrected image)
        argument: ``-B``
no_pve: (a boolean)
        turn off PVE (partial volume estimation)
        argument: ``--nopve``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
bias_lowpass: (4 <= a long integer <= 40)
        bias field smoothing extent (FWHM) in mm
        argument: ``-l %d``
init_seg_smooth: (0.0001 <= a floating point number <= 0.1)
        initial segmentation spatial smoothness (during bias field
        estimation)
        argument: ``-f %.3f``
no_bias: (a boolean)
        do not remove bias field
        argument: ``-N``
manual_seg: (an existing file name)
        Filename containing intensities
        argument: ``-s %s``
bias_iters: (1 <= a long integer <= 10)
        number of main-loop iterations during bias-field removal
        argument: ``-I %d``
probability_maps: (a boolean)
        outputs individual probability maps
        argument: ``-p``
use_priors: (a boolean)
        use priors throughout
        argument: ``-P``
number_classes: (1 <= a long integer <= 10)
        number of tissue-type classes
        argument: ``-n %d``
segments: (a boolean)
        outputs a separate binary image for each tissue type
        argument: ``-g``
hyper: (0.0 <= a floating point number <= 1.0)
        segmentation spatial smoothness
        argument: ``-H %.2f``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_GZ')
        FSL output type
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
output_biasfield: (a boolean)
        output estimated bias field
        argument: ``-b``
verbose: (a boolean)
        switch on diagnostic messages
        argument: ``-v``
mixel_smooth: (0.0 <= a floating point number <= 1.0)
        spatial smoothness for mixeltype
        argument: ``-R %.2f``
img_type: (1 or 2 or 3)
        int specifying type of image: (1 = T1, 2 = T2, 3 = PD)

```

(continues on next page)

(continued from previous page)

```
argument: ``-t %d``
```

Outputs:

```
mixeltype: (a file name)
            path/name of mixeltype volume file _mixeltype
tissue_class_map: (an existing file name)
                  path/name of binary segmented volume file one val for each class
                  _seg
partial_volume_map: (a file name)
                    path/name of partial volume file _pveseg
bias_field: (a list of items which are a file name)
probability_maps: (a list of items which are a file name)
tissue_class_files: (a list of items which are a file name)
partial_volume_files: (a list of items which are a file name)
restored_image: (a list of items which are a file name)
```

References:

None

63.8.5 FIRST[Link to code](#)

Wraps the executable command run_first_all.

FSL run_first_all wrapper for segmentation of 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
         argument: ``-i %s``, position: -2
out_file: (a file name, nipy default value: segmented)
          output data file
          argument: ``-o %s``, position: -1

[Optional]
affine_file: (an existing file name)
             Affine matrix to use (e.g. img2std.mat) (does not re-run
             registration)
             argument: ``-a %s``, position: 6
method_as_numerical_threshold: (a float)
                                Specify a numerical threshold value or use the 'method' input to
                                choose auto, fast, or none
                                argument: ``-m %.4f``, position: 4
no_cleanup: (a boolean)
```

(continues on next page)

(continued from previous page)

```

        Input structural image is already brain-extracted
        argument: ``-d``, position: 3
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
    argument: ``-s %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
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
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
    argument: ``-m %s``, position: 4
    mutually_exclusive: method_as_numerical_threshold
brain_extracted: (a boolean)
    Input structural image is already brain-extracted
    argument: ``-b``, position: 2
verbose: (a boolean)
    Use verbose logging.
    argument: ``-v``, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

```

Outputs:

```

bvars: (a list of items which are an existing file name)
    bvars for each subcortical region
segmentation_file: (an existing file name)
    4D image file containing a single volume per segmented region
original_segmentations: (an existing file name)
    3D image file containing the segmented regions as integer values.
    Uses CMA labelling
vtk_surfaces: (a list of items which are an existing file name)
    VTK format meshes for each subcortical region

```

References:

None

63.8.6 FLIRT[Link to code](#)

Wraps the executable command flirt.

FSL FLIRT wrapper for coregistration

For complete details, see the [FLIRT Documentation](#).**To print out the command line help, use:** fsl.FLIRT().inputs_help()

Examples

```

>>> from nipy.interfaces import fsl
>>> from nipy.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]
reference: (an existing file name)
    reference file
    argument: ``-ref %s``, position: 1
in_file: (an existing file name)
    input file
    argument: ``-in %s``, position: 0

[Optional]
padding_size: (an integer (int or long))
    for applyxfm: interpolates outside image by size
    argument: ``-paddingsize %d``
wmcoords: (a file name)
    white matter boundary coordinates for BBR cost function
    argument: ``-wmcoords %s``
in_matrix_file: (a file name)
    input 4x4 affine matrix
    argument: ``-init %s``
in_weight: (an existing file name)
    File for input weighting volume
    argument: ``-inweight %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
    argument: ``-searchrz %s``
wm_seg: (a file name)
    white matter segmentation volume needed by BBR cost function
    argument: ``-wmseg %s``
min_sampling: (a float)
    set minimum voxel dimension for sampling
    argument: ``-minsampling %f``
pedir: (an integer (int or long))
    phase encode direction of EPI - 1/2/3=x/y/z & -1/-2/-3=-x/-y/-z
    argument: ``-pedir %d``
ref_weight: (an existing file name)
    File for reference weighting volume
    argument: ``-refweight %s``
interp: ('trilinear' or 'nearestneighbour' or 'sinc' or 'spline')
    final interpolation method used in reslicing
    argument: ``-interp %s``
uses_qform: (a boolean)
    initialize using sform or qform
    argument: ``-usesqform``

```

(continues on next page)

(continued from previous page)

```

searchr_y: (a list of from 2 to 2 items which are an integer (int or
            long))
            search angles along y-axis, in degrees
            argument: ``-searchry %s``
cost_func: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
            'leastsq' or 'labeldiff' or 'bbr')
            cost function
            argument: ``-searchcost %s``
coarse_search: (an integer (int or long))
            coarse search delta angle
            argument: ``-coarsesearch %d``
echospat: (a float)
            value of EPI echo spacing - units of seconds
            argument: ``-echospat %f``
out_file: (a file name)
            registered output file
            argument: ``-out %s``, position: 2
dof: (an integer (int or long))
            number of transform degrees of freedom
            argument: ``-dof %d``
bbrslope: (a float)
            value of bbr slope
            argument: ``-bbrslope %f``
out_matrix_file: (a file name)
            output affine matrix in 4x4 asciiii format
            argument: ``-omat %s``, position: 3
rigid2D: (a boolean)
            use 2D rigid body mode - ignores dof
            argument: ``-2D``
display_init: (a boolean)
            display initial matrix
            argument: ``-displayinit``
no_resample_blur: (a boolean)
            do not use blurring on downsampling
            argument: ``-noresampblur``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
            'NIFTI_GZ')
            FSL output type
datatype: ('char' or 'short' or 'int' or 'float' or 'double')
            force output data type
            argument: ``-datatype %s``
sinc_width: (an integer (int or long))
            full-width in voxels
            argument: ``-sincwidth %d``
bins: (an integer (int or long))
            number of histogram bins
            argument: ``-bins %d``
apply_isoxfm: (a float)
            as applyxfm but forces isotropic resampling
            argument: ``-applyisoxfm %f``
            mutually_exclusive: apply_xfm
bbrtype: ('signed' or 'global_abs' or 'local_abs')
            type of bbr cost function: signed [default], global_abs, local_abs
            argument: ``-bbrtype %s``
cost: ('mutualinfo' or 'corratio' or 'normcorr' or 'normmi' or
            'leastsq' or 'labeldiff' or 'bbr')
            cost function

```

(continues on next page)

(continued from previous page)

```

        argument: ``-cost %s``
no_clamp: (a boolean)
    do not use intensity clamping
    argument: ``-noclamp``
no_resample: (a boolean)
    do not change input sampling
    argument: ``-noresample``
fieldmap: (a file name)
    fieldmap image in rads/s - must be already registered to the
    reference image
    argument: ``-fieldmap %s``
schedule: (an existing file name)
    replaces default schedule
    argument: ``-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
sinc_window: ('rectangular' or 'hanning' or 'blackman')
    sinc window
    argument: ``-sincwindow %s``
fine_search: (an integer (int or long))
    fine search delta angle
    argument: ``-finsearch %d``
out_log: (a file name)
    output log
    requires: save_log
force_scaling: (a boolean)
    force rescaling even for low-res images
    argument: ``-forcescaling``
searchr_x: (a list of from 2 to 2 items which are an integer (int or
    long))
    search angles along x-axis, in degrees
    argument: ``-searchrx %s``
no_search: (a boolean)
    set all angular searches to ranges 0 to 0
    argument: ``-nosearch``
apply_xfm: (a boolean)
    apply transformation supplied by in_matrix_file or uses_qform to use
    the affine matrix stored in the reference header
    argument: ``-applyxfm``
save_log: (a boolean)
    save to log file
fieldmapmask: (a file name)
    mask for fieldmap image
    argument: ``-fieldmapmask %s``
wmnorms: (a file name)
    white matter boundary normals for BBR cost function
    argument: ``-wmnorms %s``
angle_rep: ('quaternion' or 'euler')
    representation of rotation angles
    argument: ``-anglerep %s``
bgvalue: (a float)
    use specified background value for points outside FOV
    argument: ``-setbackground %f``
verbose: (an integer (int or long))
    verbose mode, 0 is least

```

(continues on next page)

(continued from previous page)

```

        argument: ``-verbose %d``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``

```

Outputs:

```

out_matrix_file: (an existing file name)
    path/name of calculated affine transform (if generated)
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)

```

References:

None

63.8.7 FNIRT[Link to code](#)Wraps the executable command `fnirt`.

FSL FNIRT wrapper 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 --out=fieldcoeff.nii --in=structural.nii --in_fwhm=8,4,2,2 --ref=mni.nii --
↪ subsamp=4,2,1,1 --warpres=6,6,6 --iout=warped.nii'

```

Inputs:

```

[Mandatory]
ref_file: (an existing file name)
    name of reference image

```

(continues on next page)

(continued from previous page)

```

        argument: ``--ref=%s``
in_file: (an existing file name)
        name of input image
        argument: ``--in=%s``

[Optional]
modulatedref_file: (a boolean or a file name)
        name of file for writing out intensity modulated --ref (for
        diagnostic purposes)
        argument: ``--refout=%s``
skip_implicit_ref_masking: (a boolean)
        skip implicit masking based on value in --ref image. Default = 0
        argument: ``--imprefm=0``
bias_regularization_lambda: (a float)
        Weight of regularisation for bias-field, default 10000
        argument: ``--biaslambda=%f``
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)
        argument: ``--applyinmask=%s``
        mutually_exclusive: skip_inmask
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
        argument: ``--warpres=%d,%d,%d``
fieldcoeff_file: (a boolean or a file name)
        name of output file with field coefficients or true
        argument: ``--cout=%s``
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]
        argument: ``--reffwhm=%s``
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]
        argument: ``--infwhm=%s``
intensity_mapping_model: ('none' or 'global_linear' or
        'global_non_linear' or 'local_linear' or
        'global_non_linear_with_bias' or 'local_non_linear')
        Model for intensity-mapping
        argument: ``--intmod=%s``
skip_implicit_in_masking: (a boolean)
        skip implicit masking based on value in --in image. Default = 0
        argument: ``--impinm=0``
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)
        argument: ``--applyrefmask=%s``
        mutually_exclusive: skip_refmask
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_GZ')
        FSL output type
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)
        argument: ``--estint=%s``
        mutually_exclusive: skip_intensity_mapping
warped_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

        name of output image
        argument: ``--iout=%s``
skip_intensity_mapping: (a boolean)
        Skip estimate intensity-mapping default false
        argument: ``--estint=0``
        mutually_exclusive: apply_intensity_mapping
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
        argument: ``--biasres=%d,%d,%d``
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]
        argument: ``--miter=%s``
refmask_file: (an existing file name)
        name of file with mask in reference space
        argument: ``--refmask=%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
        argument: ``--intin=%s``
jacobian_range: (a tuple of the form: (a float, a float))
        Allowed range of Jacobian determinants, default 0.01, 100.0
        argument: ``--jacrange=%f,%f``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
regularization_lambda: (a list of items which are a float)
        Weight of regularisation, default depending on --ssqlambda and
        --regmod switches. See user documetation.
        argument: ``--lambda=%s``
hessian_precision: ('double' or 'float')
        Precision for representing Hessian, double or float. Default double
        argument: ``--numprec=%s``
skip_inmask: (a boolean)
        skip specified inmask if set, default false
        argument: ``--applyinmask=0``
        mutually_exclusive: apply_inmask
jacobian_file: (a boolean or a file name)
        name of file for writing out the Jacobian of the field (for
        diagnostic or VBM purposes)
        argument: ``--jout=%s``
inmask_file: (an existing file name)
        name of file with mask in input image space
        argument: ``--inmask=%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
inmask_val: (a float)
        Value to mask out in --in image. Default =0.0
        argument: ``--impinval=%f``
field_file: (a boolean or a file name)
        name of output file with field or true
        argument: ``--fout=%s``
subsampling_scheme: (a list of items which are an integer (int or

```

(continues on next page)

(continued from previous page)

```

        long))
        sub-sampling scheme, list, default [4, 2, 1, 1]
        argument: ``--subsamp=%s``
regularization_model: ('membrane_energy' or 'bending_energy')
        Model for regularisation of warp-field [membrane_energy
        bending_energy], default bending_energy
        argument: ``--regmod=%s``
intensity_mapping_order: (an integer (int or long))
        Order of poynomial for mapping intensities, default 5
        argument: ``--intorder=%d``
derive_from_ref: (a boolean)
        If true, ref image is used to calculate derivatives. Default false
        argument: ``--refderiv``
affine_file: (an existing file name)
        name of file containing affine transform
        argument: ``--aff=%s``
skip_lambda_ssq: (a boolean)
        If true, lambda is not weighted by current ssq, default false
        argument: ``--ssqlambda=0``
refmask_val: (a float)
        Value to mask out in --ref image. Default =0.0
        argument: ``--imprefval=%f``
out_intensitymap_file: (a boolean or a file name)
        name of files for writing information pertaining to intensity
        mapping
        argument: ``--intout=%s``
spline_order: (an integer (int or long))
        Order of spline, 2->Qadratic spline, 3->Cubic spline. Default=3
        argument: ``--splineorder=%d``
skip_refmask: (a boolean)
        Skip specified refmask if set, default false
        argument: ``--applyrefmask=0``
        mutually_exclusive: apply_refmask
inwarp_file: (an existing file name)
        name of file containing initial non-linear warps
        argument: ``--inwarp=%s``
config_file: ('T1_2_MNI152_2mm' or 'FA_2_FMRIB58_1mm' or an existing
        file name)
        Name of config file specifying command line arguments
        argument: ``--config=%s``
log_file: (a file name)
        Name of log-file
        argument: ``--logout=%s``

```

Outputs:

```

modulatedref_file: (a file name)
        file containing intensity modulated --ref
warped_file: (an existing file name)
        warped image
fieldcoeff_file: (an existing file name)
        file with field coefficients
jacobian_file: (a file name)
        file containing Jacobian of the field
field_file: (a file name)
        file with warp field
log_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

        Name of log-file
out_intensitymap_file: (a list of from 2 to 2 items which are a file
                        name)
                        files containing info pertaining to intensity mapping

```

References:

None

63.8.8 FUGUE[Link to code](#)Wraps the executable command `fugue`.

FSL FUGUE set of tools for EPI distortion correction

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 --unwarmdir=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 --unwarmdir=y'
>>> fugue.run()

```

Inputs:

```

[Optional]
icorr: (a boolean)
    apply intensity correction to unwarping (pixel shift method only)
    argument: ``--icorr``
    requires: shift_in_file
median_2dfilter: (a boolean)
    apply 2D median filtering
    argument: ``--median``
save_shift: (a boolean)
    write pixel shift volume
    mutually_exclusive: save_unmasked_shift
no_gap_fill: (a boolean)
    do not apply gap-filling measure to the fieldmap
    argument: ``--nofill``
asym_se_time: (a float)
    set the fieldmap asymmetric spin echo time (sec)
    argument: ``--asym=%10f``
save_unmasked_shift: (a boolean)
    saves the unmasked shiftmap when using --saveshift
    argument: ``--unmaskshift``
    mutually_exclusive: save_shift
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
dwell_time: (a float)
    set the EPI dwell time per phase-encode line - same as echo spacing
    - (sec)
    argument: ``--dwell=%10f``
save_unmasked_fmap: (a boolean)
    saves the unmasked fieldmap when using --savefmap
    argument: ``--unmaskfmap``
    mutually_exclusive: save_fmap
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
    argument: ``--fourier=%d``
nokspace: (a boolean)
    do not use k-space forward warping
    argument: ``--nokspace``
despike_2dfilter: (a boolean)
    apply a 2D de-spiking filter

```

(continues on next page)

(continued from previous page)

```

        argument: ``--despike``
shift_out_file: (a file name)
    filename for saving pixel shift volume
    argument: ``--saveshift=%s``
phasesmap_in_file: (an existing file name)
    filename for input phase image
    argument: ``--phasesmap=%s``
no_extend: (a boolean)
    do not apply rigid-body extrapolation to the fieldmap
    argument: ``--noextend``
dwell_to_asym_ratio: (a float)
    set the dwell to asym time ratio
    argument: ``--dwelltoasym=%.10f``
pava: (a boolean)
    apply monotonic enforcement via PAVA
    argument: ``--pava``
unwarp_direction: ('x' or 'y' or 'z' or 'x-' or 'y-' or 'z-')
    specifies direction of warping (default y)
    argument: ``--unwarpcdir=%s``
fmap_in_file: (an existing file name)
    filename for loading fieldmap (rad/s)
    argument: ``--loadfmap=%s``
warped_file: (a file name)
    apply forward warping and save as filename
    argument: ``--warp=%s``
    mutually_exclusive: unwarp_file
    requires: in_file
phase_conjugate: (a boolean)
    apply phase conjugate method of unwarping
    argument: ``--phaseconj``
save_fmap: (a boolean)
    write field map volume
    mutually_exclusive: save_unmasked_fmap
smooth2d: (a float)
    apply 2D Gaussian smoothing of sigma N (in mm)
    argument: ``--smooth2=%.2f``
despike_threshold: (a float)
    specify the threshold for de-spiking (default=3.0)
    argument: ``--despikethreshold=%s``
smooth3d: (a float)
    apply 3D Gaussian smoothing of sigma N (in mm)
    argument: ``--smooth3=%.2f``
poly_order: (an integer (int or long))
    apply polynomial fitting of order N
    argument: ``--poly=%d``
icorr_only: (a boolean)
    apply intensity correction only
    argument: ``--icorronly``
    requires: unwarp_file
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
unwarp_file: (a file name)
    apply unwarping and save as filename
    argument: ``--unwarp=%s``
    mutually_exclusive: warped_file
    requires: in_file

```

(continues on next page)

(continued from previous page)

```
fmap_out_file: (a file name)
    filename for saving fieldmap (rad/s)
    argument: ``--savefmap=%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
shift_in_file: (an existing file name)
    filename for reading pixel shift volume
    argument: ``--loadshift=%s``
in_file: (an existing file name)
    filename of input volume
    argument: ``--in=%s``
mask_file: (an existing file name)
    filename for loading valid mask
    argument: ``--mask=%s``
```

Outputs:

```
shift_out_file: (a file name)
    voxel shift map file
warped_file: (a file name)
    forward warped file
unwarped_file: (a file name)
    unwarped file
fmap_out_file: (a file name)
    fieldmap file
```

References:

None

63.8.9 MCFLIRT[Link to code](#)Wraps the executable command `mcflirt`.

FSL MCFLIRT wrapper for 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
    argument: ``-in %s``, position: 0
```

(continues on next page)

(continued from previous page)

```

[Optional]
init: (an existing file name)
    initial transformation matrix
    argument: ``-init %s``
use_gradient: (a boolean)
    run search on gradient images
    argument: ``-gdt``
stages: (an integer (int or long))
    stages (if 4, perform final search with sinc interpolation)
    argument: ``-stages %d``
stats_imgs: (a boolean)
    produce variance and std. dev. images
    argument: ``-stats``
bins: (an integer (int or long))
    number of histogram bins
    argument: ``-bins %d``
scaling: (a float)
    scaling factor to use
    argument: ``-scaling %.2f``
save_rms: (a boolean)
    save rms displacement parameters
    argument: ``-rmsabs -rmsrel``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
use_contour: (a boolean)
    run search on contour images
    argument: ``-edge``
smooth: (a float)
    smoothing factor for the cost function
    argument: ``-smooth %.2f``
rotation: (an integer (int or long))
    scaling factor for rotation tolerances
    argument: ``-rotation %d``
cost: ('mutualinfo' or 'woods' or 'corratio' or 'normcorr' or
    'normmi' or 'leastquares')
    cost function to optimize
    argument: ``-cost %s``
dof: (an integer (int or long))
    degrees of freedom for the transformation
    argument: ``-dof %d``
interpolation: ('spline' or 'nn' or 'sinc')
    interpolation method for transformation
    argument: ``-%s_final``
ref_file: (an existing file name)
    target image for motion correction
    argument: ``-reffile %s``
out_file: (a file name)
    file to write
    argument: ``-out %s``
save_plots: (a boolean)
    save transformation parameters
    argument: ``-plots``
save_mats: (a boolean)
    save transformation matrices
    argument: ``-mats``

```

(continues on next page)

(continued from previous page)

```

mean_vol: (a boolean)
    register to mean volume
    argument: ``-meanvol``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
ref_vol: (an integer (int or long))
    volume to align frames to
    argument: ``-refvol %d``

```

Outputs:

```

mean_img: (an existing file name)
    mean timeseries image (if mean_vol=True)
variance_img: (an existing file name)
    variance image
rms_files: (a list of items which are an existing file name)
    absolute and relative displacement parameters
par_file: (an existing file name)
    text-file with motion parameters
out_file: (an existing file name)
    motion-corrected timeseries
std_img: (an existing file name)
    standard deviation image
mat_file: (a list of items which are an existing file name)
    transformation matrices

```

References:

None

63.8.10 PRELUDE[Link to code](#)

Wraps the executable command prelude.

FSL prelude wrapper for phase unwrapping

Examples

Please insert examples for use of this command

Inputs:

```

[Mandatory]
phase_file: (an existing file name)
    raw phase file
    argument: ``--phase=%s``
    mutually_exclusive: complex_phase_file
magnitude_file: (an existing file name)
    file containing magnitude image
    argument: ``--abs=%s``
    mutually_exclusive: complex_phase_file
complex_phase_file: (an existing file name)
    complex phase input volume

```

(continues on next page)

(continued from previous page)

```

        argument: ``--complex=%s``
        mutually_exclusive: magnitude_file, phase_file

[Optional]
end: (an integer (int or long))
    final image number to process (default Inf)
    argument: ``--end=%d``
labelprocess2d: (a boolean)
    does label processing in 2D (slice at a time)
    argument: ``--labelslices``
unwrapped_phase_file: (a file name)
    file containing unwrapped phase
    argument: ``--unwrap=%s``
removeramps: (a boolean)
    remove phase ramps during unwrapping
    argument: ``--removeramps``
start: (an integer (int or long))
    first image number to process (default 0)
    argument: ``--start=%d``
rawphase_file: (a file name)
    saving the raw phase output
    argument: ``--rawphase=%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
label_file: (a file name)
    saving the area labels output
    argument: ``--labels=%s``
num_partitions: (an integer (int or long))
    number of phase partitions to use
    argument: ``--numphasesplit=%d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
threshold: (a float)
    intensity threshold for masking
    argument: ``--thresh=%.10f``
process3d: (a boolean)
    forces all processing to be full 3D
    argument: ``--force3D``
    mutually_exclusive: labelprocess2d, process2d
mask_file: (an existing file name)
    filename of mask input volume
    argument: ``--mask=%s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
savemask_file: (a file name)
    saving the mask volume
    argument: ``--savemask=%s``
process2d: (a boolean)
    does all processing in 2D (slice at a time)
    argument: ``--slices``
    mutually_exclusive: labelprocess2d

```

Outputs:

```
unwrapped_phase_file: (an existing file name)
    unwrapped phase file
```

References:

None

63.8.11 SUSAN

[Link to code](#)

Wraps the executable command `susan`.

FSL SUSAN wrapper to perform smoothing

For complete details, see the [SUSAN Documenta  on](#).

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]
in_file: (an existing file name)
    filename of input timeseries
    argument: ``%s``, position: 1
brightness_threshold: (a float)
    brightness threshold and should be greater than noise level and less
    than contrast of edges to be preserved.
    argument: ``%.10f``, position: 2
fwhm: (a float)
    fwhm of smoothing, in mm, gets converted using sqrt(8*log(2))
    argument: ``%.10f``, position: 3

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
out_file: (a file name)
    output file name
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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
    argument: ``%d``, 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
```

(continues on next page)

(continued from previous page)

```

        value of class 'str', nipy default value: {})
    Environment variables
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
dimension: (3 or 2, nipy default value: 3)
        within-plane (2) or fully 3D (3)
        argument: ``%d``, position: 4

```

Outputs:

```

smoothed_file: (an existing file name)
               smoothed output file

```

References:

None

63.8.12 SliceTimer[Link to code](#)Wraps the executable command `slicetimer`.

FSL slicetimer wrapper 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
         argument: ``--in=%s``, position: 0

[Optional]
index_dir: (a boolean)
           slice indexing from top to bottom
           argument: ``--down``
global_shift: (a float)
              shift in fraction of TR, range 0:1 (default is 0.5 = no shift)
              argument: ``--tglobal``
custom_order: (an existing file name)
              filename of single-column custom interleave order file (first slice
              is referred to as 1 not 0)
              argument: ``--ocustom=%s``
out_file: (a file name)
          filename of output timeseries
          argument: ``--out=%s``

```

(continues on next page)

(continued from previous page)

```

slice_direction: (1 or 2 or 3)
    direction of slice acquisition (x=1, y=2, z=3) - default is z
    argument: ``--direction=%d``
time_repetition: (a float)
    Specify TR of data - default is 3s
    argument: ``--repeat=%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
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
custom_timings: (an existing file name)
    slice timings, in fractions of TR, range 0:1 (default is 0.5 = no
    shift)
    argument: ``--tcustom=%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
interleaved: (a boolean)
    use interleaved acquisition
    argument: ``--odd``

```

Outputs:

```

slice_time_corrected_file: (an existing file name)
    slice time corrected file

```

References:

None

63.9 interfaces.fsl.utils

63.9.1 AvScale

[Link to code](#)

Wraps the executable 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:

```

[Optional]
all_param: (a boolean)
    argument: ``--allparams``
ref_file: (an existing file name)
    reference file to get center of rotation
    argument: ``%s``, position: -1

```

(continues on next page)

(continued from previous page)

```

mat_file: (an existing file name)
    mat file to read
    argument: ``%s``, position: -2
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

rot_angles: (a list of items which are a float)
    rotation angles
skews: (a list of items which are a float)
    Skews
translations: (a list of items which are a float)
    translations
forward_half_transform: (a list of items which are a list of items
    which are a float)
    Forward Half Transform
average_scaling: (a float)
    Average Scaling
determinant: (a float)
    Determinant
backward_half_transform: (a list of items which are a list of items
    which are a float)
    Backwards Half Transform
scales: (a list of items which are a float)
    Scales (x,y,z)
rotation_translation_matrix: (a list of items which are a list of
    items which are a float)
    Rotation and Translation Matrix
left_right_orientation_preserved: (a boolean)
    True if LR orientation preserved

```

63.9.2 Complex

[Link to code](#)

Wraps the executable 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:

```

[Optional]
imaginary_out_file: (a file name)
    argument: ``%s``, position: -3
    mutually_exclusive: complex_out_file, magnitude_out_file,

```

(continues on next page)

(continued from previous page)

```

        phase_out_file, real_polar, complex_cartesian, complex_polar,
        complex_split, complex_merge
end_vol: (an integer (int or long))
    argument: ``%d``, position: -1
real_in_file: (an existing file name)
    argument: ``%s``, position: 2
complex_polar: (a boolean)
    argument: ``-complexpolar``, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
complex_split: (a boolean)
    argument: ``-complexsplit``, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
real_out_file: (a file name)
    argument: ``%s``, position: -4
    mutually_exclusive: complex_out_file, magnitude_out_file,
        phase_out_file, real_polar, complex_cartesian, complex_polar,
        complex_split, complex_merge
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
complex_in_file: (an existing file name)
    argument: ``%s``, position: 2
complex_cartesian: (a boolean)
    argument: ``-complex``, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
magnitude_in_file: (an existing file name)
    argument: ``%s``, position: 2
imaginary_in_file: (an existing file name)
    argument: ``%s``, position: 3
start_vol: (an integer (int or long))
    argument: ``%d``, position: -2
complex_out_file: (a file name)
    argument: ``%s``, position: -3
    mutually_exclusive: complex_out_file, magnitude_out_file,
        phase_out_file, real_out_file, imaginary_out_file, real_polar,
        real_cartesian
real_cartesian: (a boolean)
    argument: ``-realcartesian``, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
phase_in_file: (an existing file name)
    argument: ``%s``, position: 3
complex_merge: (a boolean)
    argument: ``-complexmerge``, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge, start_vol, end_vol
real_polar: (a boolean)
    argument: ``-realpolar``, position: 1
    mutually_exclusive: real_polar, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type

```

(continues on next page)

(continued from previous page)

```

magnitude_out_file: (a file name)
    argument: ``%s``, position: -4
    mutually_exclusive: complex_out_file, real_out_file,
        imaginary_out_file, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge
complex_in_file2: (an existing file name)
    argument: ``%s``, position: 3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
phase_out_file: (a file name)
    argument: ``%s``, position: -3
    mutually_exclusive: complex_out_file, real_out_file,
        imaginary_out_file, real_cartesian, complex_cartesian,
        complex_polar, complex_split, complex_merge

```

Outputs:

```

real_out_file: (a file name)
magnitude_out_file: (a file name)
imaginary_out_file: (a file name)
phase_out_file: (a file name)
complex_out_file: (a file name)

```

References:

None

63.9.3 ConvertWarp[Link to code](#)Wraps the executable command `convertwarp`.Use FSL [convertwarp](#) for combining multiple transforms into one.**Examples**

```

>>> from nipy.interfaces.fsl import ConvertWarp
>>> warputils = ConvertWarp()
>>> warputils.inputs.warpl = "warpfield.nii"
>>> warputils.inputs.reference = "T1.nii"
>>> warputils.inputs.relwarp = True
>>> warputils.inputs.output_type = "NIFTI_GZ"
>>> warputils.cmdline
'convertwarp --ref=T1.nii --rel --warpl=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.
    argument: ``--ref=%s``, position: 1

[Optional]
jacobian_max: (a float)
    Maximum acceptable Jacobian value for constraint (default 100.0)

```

(continues on next page)

(continued from previous page)

```

        argument: ``--jmax=%f``
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).
        argument: ``--warp2=%s``
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.
        argument: ``--out=%s``, position: -1
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.
        argument: ``--abs``
        mutually_exclusive: relwarp
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.
        argument: ``--absout``
        mutually_exclusive: out_relwarp
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.
        argument: ``--relout``
        mutually_exclusive: out_abswarp
cons_jacobian: (a boolean)
        Constrain the Jacobian of the warpfield to lie within specified
        min/max limits.
        argument: ``--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
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_GZ')
        FSL output type
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.
        argument: ``--warp1=%s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
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).
        argument: ``--shiftmap=%s``

```

(continues on next page)

(continued from previous page)

```

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.
    argument: ``--rel``
    mutually_exclusive: abswarp
premat: (an existing file name)
    filename for pre-transform (affine matrix)
    argument: ``--premat=%s``
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).
    argument: ``--postmat=%s``
jacobian_min: (a float)
    Minimum acceptable Jacobian value for constraint (default 0.01)
    argument: ``--jmin=%f``
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.
    argument: ``--shiftmdir=%s``
    requires: shift_in_file
midmat: (an existing file name)
    Name of file containing mid-warp-affine transform
    argument: ``--midmat=%s``

```

Outputs:

```

out_file: (an existing file name)
    Name of output file, containing the warp as field or coefficients.

```

References:

None

63.9.4 ConvertXFM[Link to code](#)Wraps the executable 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)

```

(continues on next page)

(continued from previous page)

```

    input transformation matrix
    argument: ``%s``, position: -1

[Optional]
concat_xfm: (a boolean)
    write joint transformation of two input matrices
    argument: ``-concat``, position: -3
    mutually_exclusive: invert_xfm, concat_xfm, fix_scale_skew
    requires: in_file2
fix_scale_skew: (a boolean)
    use secondary matrix to fix scale and skew
    argument: ``-fixscaleskew``, position: -3
    mutually_exclusive: invert_xfm, concat_xfm, fix_scale_skew
    requires: in_file2
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
    FSL output type
out_file: (a file name)
    final transformation matrix
    argument: ``-omat %s``, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
invert_xfm: (a boolean)
    invert input transformation
    argument: ``-inverse``, position: -3
    mutually_exclusive: invert_xfm, concat_xfm, fix_scale_skew
in_file2: (an existing file name)
    second input matrix (for use with fix_scale_skew or concat_xfm)
    argument: ``%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', nipyte default value: {})
    Environment variables

```

Outputs:

```

out_file: (an existing file name)
    output transformation matrix

```

References:

None

63.9.5 CopyGeom[Link to code](#)

Wraps the executable 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]
in_file: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    source image
    argument: ``%s``, position: 0
dest_file: (an existing file name)
    destination image
    argument: ``%s``, position: 1

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
ignore_dims: (a boolean)
    Do not copy image dimensions
    argument: ``-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

```

Outputs:

```

out_file: (an existing file name)
    image with new geometry header

```

References:

None

63.9.6 ExtractROI[Link to code](#)Wraps the executable 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
    argument: ``%s``, position: 0

```

(continues on next page)

(continued from previous page)

```
[Optional]
x_size: (an integer (int or long))
    argument: ``%d``, position: 3
roi_file: (a file name)
    output file
    argument: ``%s``, position: 1
t_size: (an integer (int or long))
    argument: ``%d``, position: 9
y_size: (an integer (int or long))
    argument: ``%d``, 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
y_min: (an integer (int or long))
    argument: ``%d``, position: 4
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
x_min: (an integer (int or long))
    argument: ``%d``, position: 2
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
    argument: ``%s``, position: 2
    mutually_exclusive: x_min, x_size, y_min, y_size, z_min, z_size,
        t_min, t_size
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
t_min: (an integer (int or long))
    argument: ``%d``, position: 8
z_min: (an integer (int or long))
    argument: ``%d``, position: 6
z_size: (an integer (int or long))
    argument: ``%d``, position: 7
```

Outputs:

```
roi_file: (an existing file name)
```

References:

None

63.9.7 FilterRegressor[Link to code](#)Wraps the executable 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
```

(continues on next page)

(continued from previous page)

```

        mixing matrix)
        argument: ``-d %s``, position: 3
in_file: (an existing file name)
        input file name (4D image)
        argument: ``-i %s``, position: 1
filter_all: (a boolean)
        use all columns in the design file in denoising
        argument: ``-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
        argument: ``-f %s``, position: 4
        mutually_exclusive: filter_all

[Optional]
mask: (an existing file name)
        mask image file name
        argument: ``-m %s``
out_file: (a file name)
        output file name for the filtered data
        argument: ``-o %s``, position: 2
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
        FSL output type
var_norm: (a boolean)
        perform variance-normalization on data
        argument: ``--vn``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
        Environment variables
out_vnscales: (a boolean)
        output scaling factors for variance normalization
        argument: ``--out_vnscales``

```

Outputs:

```

out_file: (an existing file name)
        output file name for the filtered data

```

References:

None

63.9.8 ImageMaths[Link to code](#)Wraps the executable command `fslmaths`.Use FSL `fslmaths` command to allow mathematical manipulation of images [FSL info](#)

Examples

```
>>> from nipy.interfaces import fsl
>>> from nipy.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)
         argument: ``%s``, position: 1

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
             FSL output type
suffix: (a unicode string)
         out_file suffix
mask_file: (an existing file name)
            use (following image>0) to mask current image
            argument: ``-mas %s``
out_file: (a file name)
           argument: ``%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)
               argument: ``-odt %s``, position: -1
in_file2: (an existing file name)
           argument: ``%s``, position: 3
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
op_string: (a unicode string)
            string defining the operation, i. e. -add
            argument: ``%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
```

Outputs:

```
out_file: (an existing file name)
```

References:

None

63.9.9 ImageMeants

[Link to code](#)

Wraps the executable 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
        argument: ``-i %s``, position: 0

[Optional]
mask: (an existing file name)
      input 3D mask
      argument: ``-m %s``
use_mm: (a boolean)
        use mm instead of voxel coordinates (for -c option)
        argument: ``--usemm``
out_file: (a file name)
          name of output text matrix
          argument: ``-o %s``
nobin: (a boolean)
        do not binarise the mask for calculation of Eigenvariates
        argument: ``--no_bin``
transpose: (a boolean)
           output results in transpose format (one row per voxel/mean)
           argument: ``--transpose``
order: (an integer (int or long), nipy default value: 1)
        select number of Eigenvariates
        argument: ``--order=%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
eig: (a boolean)
      calculate Eigenvariate(s) instead of mean (output will have 0 mean)
      argument: ``--eig``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
              FSL output type
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
show_all: (a boolean)
          show all voxel time series (within mask) instead of averaging
          argument: ``--showall``
spatial_coord: (a list of items which are an integer (int or long))
               <x y z> requested spatial coordinate (instead of mask)
               argument: ``-c %s``

```

Outputs:

```

out_file: (an existing file name)
          path/name of output text matrix

```

References:

None

63.9.10 ImageStats[Link to code](#)Wraps the executable command `fslstats`.Use FSL `fslstats` command to calculate stats from images [FSL info](#)

Examples

```
>>> from nipy.interfaces.fsl import ImageStats
>>> from nipy.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
         argument: ``%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
           argument: ``%s``, position: 3

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
             FSL output type
mask_file: (an existing file name)
            mask file used for option -k %s
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
split_4d: (a boolean)
          give a separate output line for each 3D volume of a 4D timeseries
          argument: ``-t``, position: 1
```

Outputs:

```
out_stat: (any value)
          stats output
```

References:

None

63.9.11 InvWarp

[Link to code](#)

Wraps the executable command `invwarp`.

Use FSL `Invwarp` to invert a FNIRT warp

Examples

```
>>> from nipy.interfaces.fsl import InvWarp
>>> invwarp = InvWarp()
>>> invwarp.inputs.warp = "struct2mni.nii"
```

(continues on next page)

(continued from previous page)

```
>>> 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.
    argument: ``--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).
    argument: ``--warp=%s``

[Optional]
noconstraint: (a boolean)
    Do not apply Jacobian constraint
    argument: ``--noconstraint``
regularise: (a float)
    Regularization strength (default=1.0).
    argument: ``--regularise=%f``
jacobian_max: (a float)
    Maximum acceptable Jacobian value for constraint (default 100.0)
    argument: ``--jmax=%f``
niter: (an integer (int or long))
    Determines how many iterations of the gradient-descent search that
    should be run.
    argument: ``--niter=%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
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
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.
    argument: ``--abs``
    mutually_exclusive: relative
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
jacobian_min: (a float)
    Minimum acceptable Jacobian value for constraint (default 0.01)
    argument: ``--jmin=%f``
relative: (a boolean)
```

(continues on next page)

(continued from previous page)

```

    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.
    argument: ``--rel``
    mutually_exclusive: absolute
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.
    argument: ``--out=%s``

```

Outputs:

```

inverse_warp: (an existing file name)
    Name of output file, containing warps that are the "reverse" of
    those in --warp.

```

References:

None

63.9.12 Merge[Link to code](#)Wraps the executable 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]
in_files: (a list of items which are an existing file name)
    argument: ``%s``, position: 2
dimension: ('t' or 'x' or 'y' or 'z' or 'a')
    dimension along which to merge, optionally set tr input when
    dimension is t
    argument: ``-%s``, position: 0

```

(continues on next page)

(continued from previous page)

```
[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
              FSL output type
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
tr: (a float)
    use to specify TR in seconds (default is 1.00 sec), overrides
    dimension and sets it to tr
    argument: ``%.2f``, 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
merged_file: (a file name)
             argument: ``%s``, position: 1
```

Outputs:

```
merged_file: (an existing file name)
```

References:

None

63.9.13 MotionOutliers[Link to code](#)Wraps the executable 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 '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
         argument: ``-i %s``

[Optional]
mask: (an existing file name)
      mask image for calculating metric
      argument: ``-m %s``
out_metric_values: (a file name)
                   output metric values (DVARs etc.) file name
                   argument: ``-s %s``
out_file: (a file name)
          output outlier file name
          argument: ``-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
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
```

(continues on next page)

(continued from previous page)

```

        'NIFTI_GZ')
    FSL output type
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
    argument: ``--%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
threshold: (a float)
    specify absolute threshold value (otherwise use box-plot cutoff =
    P75 + 1.5*IQR)
    argument: ``--thresh=%g``
out_metric_plot: (a file name)
    output metric values plot (DVARS etc.) file name
    argument: ``-p %s``
no_motion_correction: (a boolean)
    do not run motion correction (assumed already done)
    argument: ``--nomoco``
dummy: (an integer (int or long))
    number of dummy scans to delete (before running anything and
    creating EVs)
    argument: ``--dummy=%d``

```

Outputs:

```

out_metric_plot: (an existing file name)
out_metric_values: (an existing file name)
out_file: (an existing file name)

```

References:

None

63.9.14 Overlay[Link to code](#)Wraps the executable 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]
stat_thresh: (a tuple of the form: (a float, a float))
    min and max values for the statistical overlay
    argument: ``%.2f %.2f``, position: 7
auto_thresh_bg: (a boolean)
    automatically threshold the background image
    argument: ``-a``, position: 5
    mutually_exclusive: auto_thresh_bg, full_bg_range, bg_thresh
bg_thresh: (a tuple of the form: (a float, a float))
    min and max values for background intensity
    argument: ``%.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
    argument: ``-A``, position: 5
    mutually_exclusive: auto_thresh_bg, full_bg_range, bg_thresh
background_image: (an existing file name)
    image to use as background
    argument: ``%s``, position: 4
stat_image: (an existing file name)
    statistical image to overlay in color
    argument: ``%s``, position: 6

[Optional]
out_file: (a file name)
    combined image volume
    argument: ``%s``, position: -1
transparency: (a boolean, nipy default value: True)
    make overlay colors semi-transparent
    argument: ``%s``, position: 1
out_type: ('float' or 'int', nipy default value: float)
    write output with float or int
    argument: ``%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
stat_image2: (an existing file name)
    second statistical image to overlay in color
    argument: ``%s``, position: 9
    mutually_exclusive: show_negative_stats
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
show_negative_stats: (a boolean)
    display negative statistics in overlay
    argument: ``%s``, position: 8
    mutually_exclusive: stat_image2
stat_thresh2: (a tuple of the form: (a float, a float))
    min and max values for second statistical overlay
    argument: ``%.2f %.2f``, position: 10
use_checkerboard: (a boolean)
    use checkerboard mask for overlay
    argument: ``-c``, position: 3

```

Outputs:

```
out_file: (an existing file name)
          combined image volume
```

References:

None

63.9.15 PlotMotionParams

[Link to code](#)

Wraps the executable command `fsl_tsplo`t.

Use `fsl_tsplo`t 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]
plot_type: ('rotations' or 'translations' or 'displacement')
           which motion type to plot - rotations, translations, displacement
           argument: ``%s``
in_source: ('spm' or 'fsl')
           which program generated the motion parameter file - fsl, spm
in_file: (an existing file name or a list of items which are an
          existing file name)
          file with motion parameters
          argument: ``%s``, position: 1

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
out_file: (a file name)
          image to write
          argument: ``-o %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
plot_size: (a tuple of the form: (an integer (int or long), an
          integer (int or long)))
          plot image height and width
          argument: ``%s``
```

(continues on next page)

(continued from previous page)

```
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
Environment variables
```

Outputs:

```
out_file: (an existing file name)
          image to write
```

References:

None

63.9.16 PlotTimeSeries[Link to code](#)Wraps the executable command `fsl_tsplot`.Use `fsl_tsplot` to create images of time course plots.**Examples**

```
>>> import nipy.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
         argument: ``%s``, position: 1

[Optional]
y_min: (a float)
       minumum y value
       argument: ``--ymin=%.2f``
       mutually_exclusive: y_range
title: (a unicode string)
       plot title
       argument: ``%s``
y_range: (a tuple of the form: (a float, a float))
         min and max y axis values
         argument: ``%s``
         mutually_exclusive: y_min, y_max
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
            argument: ``%s``
            mutually_exclusive: plot_start, plot_finish
out_file: (a file name)
          image to write
```

(continues on next page)

(continued from previous page)

```

        argument: ``-o %s``
x_precision: (an integer (int or long))
    precision of x-axis labels
    argument: ``--precision=%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
labels: (a unicode string or a list of items which are a unicode
    string)
    label or list of labels
    argument: ``%s``
legend_file: (an existing file name)
    legend file
    argument: ``--legend=%s``
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
x_units: (an integer (int or long), nipy default value: 1)
    scaling units for x-axis (between 1 and length of in file)
    argument: ``-u %d``
y_max: (a float)
    maximum y value
    argument: ``--ymax=%.2f``
    mutually_exclusive: y_range
plot_finish: (an integer (int or long))
    final column from in-file to plot
    argument: ``--finish=%d``
    mutually_exclusive: plot_range
plot_size: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    plot image height and width
    argument: ``%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
plot_start: (an integer (int or long))
    first column from in-file to plot
    argument: ``--start=%d``
    mutually_exclusive: plot_range
sci_notation: (a boolean)
    switch on scientific notation
    argument: ``--sci``

```

Outputs:

```

out_file: (an existing file name)
    image to write

```

References:

None

63.9.17 PowerSpectrum[Link to code](#)Wraps the executable command `fslopspec`.

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
         argument: ``%s``, position: 0

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
             FSL output type
out_file: (a file name)
          name of output 4D file for power spectrum
          argument: ``%s``, position: 1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
out_file: (an existing file name)
          path/name of the output 4D power spectrum file
```

References:

None

63.9.18 Reorient2Std

[Link to code](#)

Wraps the executable 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)
```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
out_file: (a file name)
        argument: ``%s``
args: (a unicode string)
      Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

out_file: (an existing file name)

```

References:

None

63.9.19 RobustFOV[Link to code](#)Wraps the executable 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
        argument: ``-i %s``, position: 0

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
args: (a unicode string)
      Additional parameters to the command
        argument: ``%s``
out_transform: (a file name)
              Transformation matrix in_file to out_roi output name
        argument: ``-m %s``
brainsize: (an integer (int or long))
           size of brain in z-dimension (default 170mm/150mm)
        argument: ``-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
out_roi: (a file name)
        ROI volume output name
        argument: ``-r %s``

```

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

63.9.20 SigLoss

[Link to code](#)

Wraps the executable 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
         argument: ``-i %s``

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
            FSL output type
mask_file: (an existing file name)
            brain mask file
            argument: ``-m %s``
echo_time: (a float)
            echo time in seconds
            argument: ``--te=%f``
out_file: (a file name)
            output signal loss estimate file
            argument: ``-s %s``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
slice_direction: ('x' or 'y' or 'z')
                 slicing direction
                 argument: ``-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

```

Outputs:

```
out_file: (an existing file name)
          signal loss estimate file
```

References:

None

63.9.21 Slice

[Link to code](#)

Wraps the executable command `fslslice`.

Use `fslslice` to split a 3D file into lots of 2D files (along z-axis).

Examples

```
>>> from nipy.interfaces.fsl import Slice
>>> slice = Slice()
>>> slice.inputs.in_file = 'functional.nii'
>>> slice.inputs.out_base_name = 'sl'
>>> slice.cmdline
'fslslice functional.nii sl'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input filename
         argument: ``%s``, position: 0

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
             FSL output type
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
out_base_name: (a unicode string)
                outputs prefix
                argument: ``%s``, position: 1
```

Outputs:

```
out_files: (a list of items which are an existing file name)
```

References:

None

63.9.22 Slicer

[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: 1

[Optional]
dither_edges: (a boolean)
              produce semi-transparent (dithered) edges
              argument: ``-t``, position: 7
colour_map: (an existing file name)
            use different colour map from that stored in nifti header
            argument: ``-l %s``, position: 4
sample_axial: (an integer (int or long))
              output every n axial slices into one picture
              argument: ``-S %d``, position: 10
              mutually_exclusive: single_slice, middle_slices, all_axial,
              sample_axial
              requires: image_width
out_file: (a file name)
          picture to write
          argument: ``%s``, position: -1
scaling: (a float)
         image scale
         argument: ``-s %f``, position: 0
image_width: (an integer (int or long))
            max picture width
            argument: ``%d``, position: -2
middle_slices: (a boolean)
              output picture of mid-sagittal, axial, and coronal slices
              argument: ``-a``, position: 10
              mutually_exclusive: single_slice, middle_slices, all_axial,
              sample_axial
environ: (a dictionary with keys which are a 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
nearest_neighbour: (a boolean)
                  use nearest neighbor interpolation for output
                  argument: ``-n``, position: 8
label_slices: (a boolean, nipyre default value: True)
              display slice number
              argument: ``-L``, position: 3
image_edges: (an existing file name)
            volume to display edge overlay for (useful for checking registration)
            argument: ``%s``, position: 2
slice_number: (an integer (int or long))
```

(continues on next page)

(continued from previous page)

```

        slice number to save in picture
        argument: ``-%d``, position: 11
single_slice: ('x' or 'y' or 'z')
        output picture of single slice in the x, y, or z plane
        argument: ``-%s``, position: 10
        mutually_exclusive: single_slice, middle_slices, all_axial,
        sample_axial
        requires: slice_number
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
intensity_range: (a tuple of the form: (a float, a float))
        min and max intensities to display
        argument: ``-i %.3f %.3f``, position: 5
show_orientation: (a boolean, nipy default value: True)
        label left-right orientation
        argument: ``%s``, position: 9
all_axial: (a boolean)
        output all axial slices into one picture
        argument: ``-A``, position: 10
        mutually_exclusive: single_slice, middle_slices, all_axial,
        sample_axial
        requires: image_width
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
        'NIFTI_GZ')
        FSL output type
threshold_edges: (a float)
        use threshold for edges
        argument: ``-e %.3f``, position: 6

```

Outputs:

```

out_file: (an existing file name)
        picture to write

```

References:

None

63.9.23 Smooth[Link to code](#)Wraps the executable 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 nipy.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]
in_file: (an existing file name)
         argument: ``%s``, position: 0
sigma: (a float)
        gaussian kernel sigma in mm (not voxels)
        argument: ``-kernel gauss %.03f -fmean``, position: 1
        mutually_exclusive: fwhm
fwhm: (a float)
       gaussian kernel fwhm, will be converted to sigma in mm (not voxels)
       argument: ``-kernel gauss %.03f -fmean``, position: 1
       mutually_exclusive: sigma

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
              'NIFTI_GZ')
             FSL output type
smoothed_file: (a file name)
               argument: ``%s``, position: 2
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
```

Outputs:

```
smoothed_file: (an existing file name)
```

References:

None

63.9.24 Split

[Link to code](#)

Wraps the executable command `fslsplit`.

Uses FSL `Fslsplit` command to separate a volume into images in time, x, y or z dimension.

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input filename
    argument: ``%s``, position: 0
dimension: ('t' or 'x' or 'y' or 'z')
    dimension along which the file will be split
    argument: ``-%s``, position: 2

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
    FSL output type
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
out_base_name: (a unicode string)
    outputs prefix
    argument: ``%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
```

Outputs:

```
out_files: (a list of items which are an existing file name)
```

References:

None

63.9.25 SwapDimensions

[Link to code](#)

Wraps the executable command `fsllswapdim`.

Use `fsllswapdim` 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-style dimension codes (RL, LR, AP, PA, IS, SI).

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input image
    argument: ``%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
    argument: ``%s %s %s``

[Optional]
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
    'NIFTI_GZ')
```

(continues on next page)

(continued from previous page)

```

    FSL output type
out_file: (a file name)
    image to write
    argument: ``%s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

out_file: (an existing file name)
    image with new dimensions

```

References:

None

63.9.26 WarpPoints[Link to code](#)Wraps the executable 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]
in_coords: (an existing file name)
    filename of file containing coordinates
    argument: ``%s``, position: -1
src_file: (an existing file name)
    filename of source image
    argument: ``-src %s``
dest_file: (an existing file name)
    filename of destination image
    argument: ``-dest %s``

```

(continues on next page)

(continued from previous page)

```
[Optional]
warp_file: (an existing file name)
    filename of warpfield (e.g. intermediate2dest_warp.nii.gz)
    argument: ``-warp %s``
    mutually_exclusive: xfm_file
out_file: (a file name)
    output file name
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
coord_vox: (a boolean)
    all coordinates in voxels - default
    argument: ``-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
xfm_file: (an existing file name)
    filename of affine transform (e.g. source2dest.mat)
    argument: ``-xfm %s``
    mutually_exclusive: warp_file
coord_mm: (a boolean)
    all coordinates in mm
    argument: ``-mm``
    mutually_exclusive: coord_vox
```

Outputs:

```
out_file: (an existing file name)
    Name of output file, containing the warp as field or coefficients.
```

63.9.27 WarpPointsFromStd[Link to code](#)

Wraps the executable command `std2imgcoord`.

Use FSL `std2imgcoord` to transform point sets to standard space coordinates. Accepts plain text coordinates files.

Examples

```
>>> from nipy.interfaces.fsl import WarpPointsFromStd
>>> warppoints = WarpPointsFromStd()
>>> 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
'std2imgcoord -mm -img T1.nii -std mni.nii -warp warpfield.nii surf.txt'
>>> res = warppoints.run()
```

Inputs:

```
[Mandatory]
std_file: (an existing file name)
```

(continues on next page)

(continued from previous page)

```

        filename of the image in standard space
        argument: ``-std %s``
in_coords: (an existing file name)
        filename of file containing coordinates
        argument: ``%s``, position: -2
img_file: (an existing file name)
        filename of a destination image
        argument: ``-img %s``

[Optional]
warp_file: (an existing file name)
        filename of warpfield (e.g. intermediate2dest_warp.nii.gz)
        argument: ``-warp %s``
        mutually_exclusive: xfm_file
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
coord_vox: (a boolean)
        all coordinates in voxels - default
        argument: ``-vox``
        mutually_exclusive: coord_mm
coord_mm: (a boolean)
        all coordinates in mm
        argument: ``-mm``
        mutually_exclusive: coord_vox
xfm_file: (an existing file name)
        filename of affine transform (e.g. source2dest.mat)
        argument: ``-xfm %s``
        mutually_exclusive: warp_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', nipype default value: {})
        Environment variables

```

Outputs:

```

out_file: (an existing file name)
        Name of output file, containing the warp as field or coefficients.

```

63.9.28 WarpPointsToStd

[Link to code](#)

Wraps the executable 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 nipype.interfaces.fsl import WarpPointsToStd
>>> warppoints = WarpPointsToStd()
>>> warppoints.inputs.in_coords = 'surf.txt'
>>> warppoints.inputs.img_file = 'T1.nii'

```

(continues on next page)

(continued from previous page)

```

>>> 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]
std_file: (an existing file name)
    filename of destination image
    argument: ``-std %s``
in_coords: (an existing file name)
    filename of file containing coordinates
    argument: ``%s``, position: -1
img_file: (an existing file name)
    filename of input image
    argument: ``-img %s``

[Optional]
xfm_file: (an existing file name)
    filename of affine transform (e.g. source2dest.mat)
    argument: ``-xfm %s``
    mutually_exclusive: warp_file
premat_file: (an existing file name)
    filename of pre-warp affine transform (e.g.
    example_func2highres.mat)
    argument: ``-premat %s``
out_file: (a file name)
    output file name
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
coord_vox: (a boolean)
    all coordinates in voxels - default
    argument: ``-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
warp_file: (an existing file name)
    filename of warpfield (e.g. intermediate2dest_warp.nii.gz)
    argument: ``-warp %s``
    mutually_exclusive: xfm_file
coord_mm: (a boolean)
    all coordinates in mm
    argument: ``-mm``
    mutually_exclusive: coord_vox

```

Outputs:

```

out_file: (an existing file name)
    Name of output file, containing the warp as field or coefficients.

```

63.9.29 WarpUtils

[Link to code](#)

Wraps the executable 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]
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.
    argument: ``--ref=%s``
write_jacobian: (a boolean, nipy default value: False)
    Switch on --jac flag with automatically generated filename
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).
    argument: ``--in=%s``

[Optional]
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.
    argument: ``--out=%s``, position: -1
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).
    argument: ``--withaff``
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.
    argument: ``--outformat=%s``
warp_resolution: (a tuple of the form: (a float, a float, a float))
    Specifies the resolution/knot-spacing of the splines pertaining to
```

(continues on next page)

(continued from previous page)

```

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.
argument: ``--warpres=%0.4f,%0.4f,%0.4f``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
output_type: ('NIFTI_PAIR_GZ' or 'NIFTI_PAIR' or 'NIFTI' or
            'NIFTI_GZ')
            FSL output type
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
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.
        argument: ``--knotspace=%d,%d,%d``
out_jacobian: (a file name)
        Specifies that a (3D) file of Jacobian determinants corresponding to
        --in should be produced and written to filename.
        argument: ``--jac=%s``

```

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

64.1 interfaces.minc.base

64.1.1 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'
```

64.2 interfaces.minc.minc

64.2.1 Average

[Link to code](#)

Wraps the executable 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.
    argument: ``-filelist %s``
    mutually_exclusive: input_files, filelist
input_files: (a list of items which are a file name)
    input file(s)
    argument: ``%s``, position: -2
    mutually_exclusive: input_files, filelist

[Optional]
binarize: (a boolean)
    Binarize the volume by looking for values in a given range.
    argument: ``-binarize``
format_short: (a boolean)
    Write out short integer data.
    argument: ``-short``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
clobber: (a boolean, nipyte default value: True)
    Overwrite existing file.
    argument: ``-clobber``
nonnormalize: (a boolean)
    Do not normalize data sets (default).
    argument: ``-nonnormalize``
    mutually_exclusive: normalize, nonnormalize
avgdim: (a unicode string)
    Specify a dimension along which we wish to average.
    argument: ``-avgdim %s``
environ: (a dictionary with keys which are a 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
no_check_dimensions: (a boolean)
    Do not check dimension info.
    argument: ``-nocheck_dimensions``
    mutually_exclusive: check_dimensions, no_check_dimensions
two: (a boolean)
    Create a MINC 2 output file.
    argument: ``-2``
max_buffer_size_in_kb: (a long integer >= 0, nipyte default value:
    4096)
    Specify the maximum size of the internal buffers (in kbytes).
    argument: ``-max_buffer_size_in_kb %d``
binvalue: (a float)
    Specify a target value (+/- 0.5) for binarization. Default value:
    -1.79769e+308
    argument: ``-binvalue %s``
format_long: (a boolean)
    Superseded by -int.
    argument: ``-long``
```

(continues on next page)

(continued from previous page)

```

    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).
    argument: ``-unsigned``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
format_byte: (a boolean)
    Write out byte data.
    argument: ``-byte``
    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.
    argument: ``-signed``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
binrange: (a tuple of the form: (a float, a float))
    Specify a range for binarization. Default value: 1.79769e+308
    -1.79769e+308.
    argument: ``-binrange %s %s``
verbose: (a boolean)
    Print out log messages (default).
    argument: ``-verbose``
    mutually_exclusive: verbose, quiet
quiet: (a boolean)
    Do not print out log messages.
    argument: ``-quiet``
    mutually_exclusive: verbose, quiet
check_dimensions: (a boolean)
    Check that dimension info matches across files (default).
    argument: ``-check_dimensions``
    mutually_exclusive: check_dimensions, no_check_dimensions
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
format_float: (a boolean)
    Write out single-precision floating-point data.
    argument: ``-float``
    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.
    argument: ``-double``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
copy_header: (a boolean)
    Copy all of the header from the first file (default for one file).
    argument: ``-copy_header``
    mutually_exclusive: copy_header, no_copy_header
no_copy_header: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        Do not copy all of the header from the first file (default for many
        files)).
        argument: ``-nocopy_header``
        mutually_exclusive: copy_header, no_copy_header
normalize: (a boolean)
        Normalize data sets for mean intensity.
        argument: ``-normalize``
        mutually_exclusive: normalize, nonnormalize
voxel_range: (a tuple of the form: (an integer (int or long), an
        integer (int or long)))
        Valid range for output data.
        argument: ``-range %d %d``
weights: (a list of items which are a unicode string)
        Specify weights for averaging ("<w1>,<w2>,...").
        argument: ``-weights %s``
debug: (a boolean)
        Print out debugging messages.
        argument: ``-debug``
format_int: (a boolean)
        Write out 32-bit integer data.
        argument: ``-int``
        mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
width_weighted: (a boolean)
        Weight by dimension widths when -avgdim is used.
        argument: ``-width_weighted``
        requires: avgdim
format_filetype: (a boolean)
        Use data type of first file (default).
        argument: ``-filetype``
        mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
sdfile: (a file name)
        Specify an output sd file (default=None).
        argument: ``-sdfile %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``

```

Outputs:

```

output_file: (an existing file name)
        output file

```

64.2.2 BBox[Link to code](#)

Wraps the executable command mincbbox.

Determine a bounding box of image.

Examples

```

>>> from nipy.interfaces.minc import BBox
>>> from nipy.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
            argument: ``%s``, position: -2

[Optional]
threshold: (an integer (int or long))
            VIO_Real value threshold for bounding box. Default value: 0.
            argument: ``-threshold``

format_minreshape: (a boolean)
                   Output format for minreshape: (-start x,y,z -count dx,dy,dz
                   argument: ``-minreshape``

out_file: (a file name)
           argument: ``> %s``, position: -1

format_minccrop: (a boolean)
                 Output format for minccrop: (-xlim x1 x2 -ylim y1 y2 -zlim z1 z2
                 argument: ``-minccrop``

output_file: (a file name)
              output file containing bounding box corners

two_lines: (a boolean)
            Output on two lines: start_x y z
            width_x y z
            argument: ``-two_lines``
            mutually_exclusive: one_line, two_lines

format_minresample: (a boolean)
                    Output format for minresample: (-step x y z -start x y z -nelements
                    x y z
                    argument: ``-minresample``

args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``

one_line: (a boolean)
           Output on one line (default): start_x y z width_x y z
           argument: ``-one_line``
           mutually_exclusive: one_line, two_lines

environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
```

Outputs:

```
output_file: (an existing file name)
             output file containing bounding box corners
```

64.2.3 Beast

[Link to code](#)

Wraps the executable command mincbeast.

Extract brain image using BEaST (Brain Extraction using non-local Segmentation Technique).

Examples

```
>>> from nipyne.interfaces.minc import Beast
>>> from nipyne.interfaces.minc.testdata import nonempty_minc_data
```

```
>>> file0 = nonempty_minc_data(0)
>>> beast = Beast(input_file=file0)
>>> beast .run()
```

Inputs:

```
[Mandatory]
library_dir: (a directory name)
    library directory
    argument: ``%s``, position: -3
input_file: (a file name)
    input file
    argument: ``%s``, position: -2

[Optional]
configuration_file: (a file name)
    Specify configuration file.
    argument: ``-configuration %s``
load_moments: (a boolean)
    Do not calculate moments instead use precalculatedlibrary moments.
    (for optimization purposes)
    argument: ``-load_moments``
fill_holes: (a boolean)
    Fill holes in the binary output.
    argument: ``-fill``
clobber: (a boolean, nipyne default value: True)
    Overwrite existing file.
    argument: ``-clobber``
probability_map: (a boolean)
    Output the probability map instead of crisp mask.
    argument: ``-probability``
environ: (a dictionary with keys which are a 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
threshold_patch_selection: (a float, nipyne default value: 0.95)
    Specify threshold for patch selection. Default value: 0.95
    argument: ``-threshold %s``
patch_size: (an integer (int or long), nipyne default value: 1)
    Specify patch size for single scale approach. Default value: 1.
    argument: ``-patch_size %s``
voxel_size: (an integer (int or long), nipyne default value: 4)
    Specify voxel size for calculations (4, 2, or 1).Default value: 4.
    Assumes no multiscale. Use configurationfile for multiscale.
    argument: ``-voxel_size %s``
confidence_level_alpha: (a float, nipyne default value: 0.5)
    Specify confidence level Alpha. Default value: 0.5
    argument: ``-alpha %s``
abspath: (a boolean, nipyne default value: True)
    File paths in the library are absolute (default is relative to
    library root).
    argument: ``-abspath``
output_file: (a file name)
```

(continues on next page)

(continued from previous page)

```

        output file
        argument: ``%s``, position: -1
search_area: (an integer (int or long), nipyne default value: 2)
    Specify size of search area for single scale approach. Default
    value: 2.
    argument: ``-search_area %s``
flip_images: (a boolean)
    Flip images around the mid-sagittal plane to increase patch count.
    argument: ``-flip``
median_filter: (a boolean)
    Apply a median filter on the probability map.
    argument: ``-median``
smoothness_factor_beta: (a float, nipyne default value: 0.5)
    Specify smoothness factor Beta. Default value: 0.25
    argument: ``-beta %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
number_selected_images: (an integer (int or long), nipyne default
    value: 20)
    Specify number of selected images. Default value: 20
    argument: ``-selection_num %s``
nlm_filter: (a boolean)
    Apply an NLM filter on the probability map (experimental).
    argument: ``-nlm_filter``
same_resolution: (a boolean)
    Output final mask with the same resolution as input file.
    argument: ``-same_resolution``

```

Outputs:

```

output_file: (an existing file name)
    output mask file

```

64.2.4 BestLinReg

[Link to code](#)

Wraps the executable 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 nipyne.interfaces.minc import BestLinReg
>>> from nipyne.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
        argument: ``%s``, position: -4
target: (an existing file name)
        target Minc file
        argument: ``%s``, position: -3

[Optional]
verbose: (a boolean)
        Print out log messages. Default: False.
        argument: ``-verbose``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
output_xfm: (a file name)
        output xfm file
        argument: ``%s``, position: -2
output_mnc: (a file name)
        output mnc file
        argument: ``%s``, position: -1
clobber: (a boolean, nipy default value: True)
        Overwrite existing file.
        argument: ``-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

```

Outputs:

```

output_mnc: (an existing file name)
        output mnc file
output_xfm: (an existing file name)
        output xfm file

```

64.2.5 BigAverage[Link to code](#)

Wraps the executable 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)
    argument: ``%s``, position: -2

[Optional]
output_float: (a boolean)
    Output files with float precision.
    argument: ``--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.
    argument: ``--robust``
tmpdir: (a directory name)
    temporary files directory
    argument: ``-tmpdir %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``--verbose``
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``--clobber``
sd_file: (a file name)
    Place standard deviation image in specified file.
    argument: ``--sdfile %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
sd_file: (an existing file name)
    standard deviation image
output_file: (an existing file name)
    output file
```

64.2.6 Blob

[Link to code](#)

Wraps the executable 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
            argument: ``%s``, position: -2

[Optional]
trace: (a boolean)
       compute the trace (approximate growth and shrinkage) -- FAST
       argument: ``-trace``
magnitude: (a boolean)
           compute the magnitude of the displacement vector
           argument: ``-magnitude``
determinant: (a boolean)
             compute the determinant (exact growth and shrinkage) -- SLOW
             argument: ``-determinant``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
output_file: (a file name)
            output file
            argument: ``%s``, position: -1
translation: (a boolean)
             compute translation (structure displacement)
             argument: ``-translation``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
output_file: (an existing file name)
            output file
```

64.2.7 Blur

[Link to code](#)

Wraps the executable 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]
standard_dev: (a float)
    Standard deviation of gaussian kernel. Default value: 0.
    argument: ``-standarddev %s``
    mutually_exclusive: fwhm, fwhm3d, standard_dev
input_file: (an existing file name)
    input file
    argument: ``%s``, position: -2
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.
    argument: ``-3dfwhm %s %s %s``
    mutually_exclusive: fwhm, fwhm3d, standard_dev
fwhm: (a float)
    Full-width-half-maximum of gaussian kernel. Default value: 0.
    argument: ``-fwhm %s``
    mutually_exclusive: fwhm, fwhm3d, standard_dev

[Optional]
gradient: (a boolean)
    Create the gradient magnitude volume as well.
    argument: ``-gradient``
dimensions: (3 or 1 or 2)
    Number of dimensions to blur (either 1,2 or 3). Default value: 3.
    argument: ``-dimensions %s``
no_apodize: (a boolean)
    Do not apodize the data before blurring.
    argument: ``-no_apodize``
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``-clobber``
```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a bytes or None or a 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).
         argument: ``-gaussian``
         mutually_exclusive: gaussian, rect
output_file_base: (a file name)
         output file base
         argument: ``%s``, position: -1
partial: (a boolean)
         Create the partial derivative and gradient magnitude volumes as
         well.
         argument: ``-partial``
args: (a unicode string)
         Additional parameters to the command
         argument: ``%s``
rect: (a boolean)
         Use a rect (box) smoothing kernel.
         argument: ``-rect``
         mutually_exclusive: gaussian, rect

```

Outputs:

```

partial_dz: (a file name)
         Partial gradient dz.
partial_dx: (a file name)
         Partial gradient dx.
gradient_dxyz: (a file name)
         Gradient dxyz.
partial_dy: (a file name)
         Partial gradient dy.
partial_dxyz: (a file name)
         Partial gradient dxyz.
output_file: (an existing file name)
         Blurred output file.

```

64.2.8 Calc[Link to code](#)

Wraps the executable 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]
expression: (a unicode string)
    Expression to use in calculations.
    argument: ``-expression '%s'``
    mutually_exclusive: expression, expfile
input_files: (a list of items which are a file name)
    input file(s) for calculation
    argument: ``%s``, position: -2
filelist: (a file name)
    Specify the name of a file containing input file names.
    argument: ``-filelist %s``
    mutually_exclusive: input_files, filelist
expfile: (a file name)
    Name of file containing expression.
    argument: ``-expfile %s``
    mutually_exclusive: expression, expfile

[Optional]
eval_width: (an integer (int or long))
    Number of voxels to evaluate simultaneously.
    argument: ``-eval_width %s``
format_byte: (a boolean)
    Write out byte data.
    argument: ``-byte``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
output_illegal: (a boolean)
    Value to write out when an illegal operation is done. Default value:
    1.79769e+308
    argument: ``-illegal_value``
    mutually_exclusive: output_nan, output_zero, output_illegal_value
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``-clobber``
propagate_nan: (a boolean)
    Invalid data in any file at a voxel produces a NaN (default).
    argument: ``-propagate_nan``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
no_check_dimensions: (a boolean)
    Do not check that files have matching dimensions.
    argument: ``-nocheck_dimensions``
    mutually_exclusive: check_dimensions, no_check_dimensions
two: (a boolean)
    Create a MINC 2 output file.
    argument: ``-2``
max_buffer_size_in_kb: (a long integer >= 0)
    Specify the maximum size of the internal buffers (in kbytes).
    argument: ``-max_buffer_size_in_kb %d``
ignore_nan: (a boolean)
    Ignore invalid data (NaN) for accumulations.
    argument: ``-ignore_nan``
format_long: (a boolean)
    Superseded by -int.

```

(continues on next page)

(continued from previous page)

```

    argument: ``-long``
    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).
    argument: ``-unsigned``
    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.
    argument: ``-signed``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
quiet: (a boolean)
    Do not print out log messages.
    argument: ``-quiet``
    mutually_exclusive: verbose, quiet
check_dimensions: (a boolean)
    Check that files have matching dimensions (default).
    argument: ``-check_dimensions``
    mutually_exclusive: check_dimensions, no_check_dimensions
format_short: (a boolean)
    Write out short integer data.
    argument: ``-short``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
format_float: (a boolean)
    Write out single-precision floating-point data.
    argument: ``-float``
    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.
    argument: ``-double``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
copy_header: (a boolean)
    Copy all of the header from the first file.
    argument: ``-copy_header``
    mutually_exclusive: copy_header, no_copy_header
no_copy_header: (a boolean)
    Do not copy all of the header from the first file.
    argument: ``-nocopy_header``
    mutually_exclusive: copy_header, no_copy_header
output_zero: (a boolean)
    Output zero when an illegal operation is done.
    argument: ``-zero``
    mutually_exclusive: output_nan, output_zero, output_illegal_value

```

(continues on next page)

(continued from previous page)

```

voxel_range: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    Valid range for output data.
    argument: ``-range %d %d``
output_nan: (a boolean)
    Output NaN when an illegal operation is done (default).
    argument: ``-nan``
    mutually_exclusive: output_nan, output_zero, output_illegal_value
debug: (a boolean)
    Print out debugging messages.
    argument: ``-debug``
format_int: (a boolean)
    Write out 32-bit integer data.
    argument: ``-int``
    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).
    argument: ``-filetype``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
verbose: (a boolean)
    Print out log messages (default).
    argument: ``-verbose``
    mutually_exclusive: verbose, quiet
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outfiles: (a list of items which are a tuple of the form: (a unicode
    string, a file name))

```

Outputs:

```

output_file: (an existing file name)
    output file

```

64.2.9 Convert[Link to code](#)

Wraps the executable 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)

```

(continues on next page)

(continued from previous page)

```

    input file for converting
    argument: ``%s``, position: -2

[Optional]
two: (a boolean)
    Create a MINC 2 output file.
    argument: ``-2``
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.
    argument: ``-template``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
chunk: (a long integer >= 0)
    Set the target block size for chunking (0 default, >1 block size).
    argument: ``-chunk %d``
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``-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).
    argument: ``-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

```

Outputs:

```

output_file: (an existing file name)
    output file

```

64.2.10 Copy[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -2

[Optional]
pixel_values: (a boolean)
    Copy pixel values as is.
    argument: ``-pixel_values``
mutually_exclusive: pixel_values, real_values

```

(continues on next page)

(continued from previous page)

```

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
real_values: (a boolean)
    Copy real pixel intensities (default).
    argument: ``-real_values``
    mutually_exclusive: pixel_values, real_values
environ: (a dictionary with keys which are a 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

```

Outputs:

```

output_file: (an existing file name)
    output file

```

64.2.11 Dump

[Link to code](#)

Wraps the executable command mincdump.

Dump a MINC file. Typically used in conjunction with mincgen (see Gen).

Examples

```

>>> from nipyne.interfaces.minc import Dump
>>> from nipyne.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
    argument: ``%s``, position: -2

[Optional]
header_data: (a boolean)
    Header information only, no data.
    argument: ``-h``
    mutually_exclusive: coordinate_data, header_data
line_length: (a long integer >= 0)
    Line length maximum in data section (default 80).
    argument: ``-l %d``
netcdf_name: (a unicode string)
    Name for netCDF (default derived from file name).

```

(continues on next page)

(continued from previous page)

```

    argument: ``-n %s``
out_file: (a file name)
    argument: ``> %s``, position: -1
output_file: (a file name)
    output file
annotations_brief: ('c' or 'f')
    Brief annotations for C or Fortran indices in data.
    argument: ``-b %s``
    mutually_exclusive: annotations_brief, annotations_full
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
annotations_full: ('c' or 'f')
    Full annotations for C or Fortran indices in data.
    argument: ``-f %s``
    mutually_exclusive: annotations_brief, annotations_full
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
    argument: ``%s``
variables: (a list of items which are a unicode string)
    Output data for specified variables only.
    argument: ``-v %s``
coordinate_data: (a boolean)
    Coordinate variable data and header information.
    argument: ``-c``
    mutually_exclusive: coordinate_data, header_data
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

```

Outputs:

```

output_file: (an existing file name)
    output file

```

64.2.12 Extract[Link to code](#)

Wraps the executable command mincextract.

Dump a hyperslab of MINC file data.

Examples

```

>>> from nipy.interfaces.minc import Extract
>>> from nipy.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
    argument: ``%s``, position: -2

[Optional]
count: (a list of items which are an integer (int or long))
    Specifies edge lengths of hyperslab to read.
    argument: ``-count %s``
write_unsigned: (a boolean)
    Write out unsigned data.
    argument: ``-unsigned``
    mutually_exclusive: write_signed, write_unsigned
image_minimum: (a float)
    Specify the minimum real image value for normalization.Default
    value: 1.79769e+308.
    argument: ``-image_minimum %s``
start: (a list of items which are an integer (int or long))
    Specifies corner of hyperslab (C conventions for indices).
    argument: ``-start %s``
write_int: (a boolean)
    Write out data as 32-bit integers.
    argument: ``-int``
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
nonnormalize: (a boolean)
    Turn off pixel normalization.
    argument: ``-nonnormalize``
    mutually_exclusive: normalize, nonnormalize
flip_negative_direction: (a boolean)
    Flip images to always have negative direction.
    argument: ``-negative_direction``
    mutually_exclusive: flip_positive_direction,
        flip_negative_direction, flip_any_direction
write_double: (a boolean)
    Write out data as double precision floating-point values.
    argument: ``-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.
    argument: ``-float``
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
image_maximum: (a float)
    Specify the maximum real image value for normalization.Default
    value: 1.79769e+308.
    argument: ``-image_maximum %s``
flip_x_positive: (a boolean)
    Flip images to give positive xspace:step value (left-to-right).

```

(continues on next page)

(continued from previous page)

```

        argument: ``+xdirection``
        mutually_exclusive: flip_x_positive, flip_x_negative, flip_x_any
flip_z_any: (a boolean)
    Don't flip images along z-axis (default).
    argument: ``-zanydirection``
    mutually_exclusive: flip_z_positive, flip_z_negative, flip_z_any
flip_y_negative: (a boolean)
    Flip images to give negative yspace:step value (ant-to-post).
    argument: ``-ydirection``
    mutually_exclusive: flip_y_positive, flip_y_negative, flip_y_any
write_signed: (a boolean)
    Write out signed data.
    argument: ``-signed``
    mutually_exclusive: write_signed, write_unsigned
flip_x_any: (a boolean)
    Don't flip images along x-axis (default).
    argument: ``-xanydirection``
    mutually_exclusive: flip_x_positive, flip_x_negative, flip_x_any
out_file: (a file name)
    argument: ``> %s``, position: -1
flip_y_positive: (a boolean)
    Flip images to give positive yspace:step value (post-to-ant).
    argument: ``+ydirection``
    mutually_exclusive: flip_y_positive, flip_y_negative, flip_y_any
output_file: (a file name)
    output file
flip_y_any: (a boolean)
    Don't flip images along y-axis (default).
    argument: ``-yanydirection``
    mutually_exclusive: flip_y_positive, flip_y_negative, flip_y_any
write_byte: (a boolean)
    Write out data as bytes.
    argument: ``-byte``
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
flip_z_negative: (a boolean)
    Flip images to give negative zspace:step value (sup-to-inf).
    argument: ``-zdirection``
    mutually_exclusive: flip_z_positive, flip_z_negative, flip_z_any
write_long: (a boolean)
    Superseded by write_int.
    argument: ``-long``
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
flip_positive_direction: (a boolean)
    Flip images to always have positive direction.
    argument: ``-positive_direction``
    mutually_exclusive: flip_positive_direction,
        flip_negative_direction, flip_any_direction
normalize: (a boolean)
    Normalize integer pixel values to file max and min.
    argument: ``-normalize``
    mutually_exclusive: normalize, nonnormalize
flip_any_direction: (a boolean)
    Do not flip images (Default).

```

(continues on next page)

(continued from previous page)

```

    argument: ``-any_direction``
    mutually_exclusive: flip_positive_direction,
        flip_negative_direction, flip_any_direction
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.
    argument: ``-range %s %s``
flip_z_positive: (a boolean)
    Flip images to give positive zspace:step value (inf-to-sup).
    argument: ``+zdirection``
    mutually_exclusive: flip_z_positive, flip_z_negative, flip_z_any
write_ascii: (a boolean)
    Write out data as ascii strings (default).
    argument: ``-ascii``
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned
image_range: (a tuple of the form: (a float, a float))
    Specify the range of real image values for normalization.
    argument: ``-image_range %s %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
flip_x_negative: (a boolean)
    Flip images to give negative xspace:step value (right-to-left).
    argument: ``-xdirection``
    mutually_exclusive: flip_x_positive, flip_x_negative, flip_x_any
write_short: (a boolean)
    Write out data as short integers.
    argument: ``-short``
    mutually_exclusive: write_ascii, write_ascii, write_byte,
        write_short, write_int, write_long, write_float, write_double,
        write_signed, write_unsigned

```

Outputs:

```

output_file: (an existing file name)
    output file in raw/text format

```

64.2.13 Gennlxfm

[Link to code](#)

Wraps the executable 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:

```
[Optional]
step: (an integer (int or long))
    Output ident xfm step [default: 1].
    argument: ``-step %s``
like: (an existing file name)
    Generate a nlxfm like this file.
    argument: ``-like %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``-verbose``
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``-clobber``
ident: (a boolean)
    Generate an identity xfm. Default: False.
    argument: ``-ident``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
output_grid: (an existing file name)
    output grid
output_file: (an existing file name)
    output file
```

64.2.14 Math[Link to code](#)

Wraps the executable 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.
    argument: ``-filelist %s``
    mutually_exclusive: input_files, filelist
input_files: (a list of items which are a file name)
    input file(s) for calculation
    argument: ``%s``, position: -2
    mutually_exclusive: input_files, filelist

[Optional]
segment: (a tuple of the form: (a float, a float))
    Segment a volume using range of -const2: within range = 1, outside
    range = 0.
    argument: ``-segment -const2 %s %s``
format_byte: (a boolean)
    Write out byte data.
    argument: ``-byte``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
minimum: (a boolean)
    Find minimum of N volumes.
    argument: ``-minimum``
calc_sub: (a boolean or a float)
    Subtract 2 volumes or volume - constant.
    argument: ``-sub``
no_check_dimensions: (a boolean)
    Do not check dimension info.
    argument: ``-nocheck_dimensions``
    mutually_exclusive: check_dimensions, no_check_dimensions
invert: (a float)
    Calculate 1/c.
    argument: ``-invert -const %s``
ignore_nan: (a boolean)
    Ignore invalid data (NaN) for accumulations.
    argument: ``-ignore_nan``
abs: (a boolean)
    Take absolute value of a volume.
    argument: ``-abs``
test_eq: (a boolean or a float)
    Test for integer vol1 == vol2 or vol1 == constant.
    argument: ``-eq``
calc_add: (a boolean or a float)
    Add N volumes or volume + constant.
    argument: ``-add``
calc_and: (a boolean)
    Calculate vol1 && vol2 (&& ...).
    argument: ``-and``
two: (a boolean)
    Create a MINC 2 output file.
    argument: ``-2``
check_dimensions: (a boolean)
    Check that dimension info matches across files (default).
    argument: ``-check_dimensions``
    mutually_exclusive: check_dimensions, no_check_dimensions
format_short: (a boolean)

```

(continues on next page)

(continued from previous page)

```

    Write out short integer data.
    argument: ``-short``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
calc_div: (a boolean or a float)
    Divide 2 volumes or volume / constant.
    argument: ``-div``
format_double: (a boolean)
    Write out double-precision floating-point data.
    argument: ``-double``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
no_copy_header: (a boolean)
    Do not copy all of the header from the first file (default for many
    files)).
    argument: ``-nocopy_header``
    mutually_exclusive: copy_header, no_copy_header
log: (a tuple of the form: (a float, a float))
    Calculate  $\log(x/c2)/c1$ . The constants  $c1$  and  $c2$  default to 1.
    argument: ``-log -const2 %s %s``
scale: (a tuple of the form: (a float, a float))
    Scale a volume:  $\text{volume} * c1 + c2$ .
    argument: ``-scale -const2 %s %s``
exp: (a tuple of the form: (a float, a float))
    Calculate  $c2 * \exp(c1 * x)$ . Both constants must be specified.
    argument: ``-exp -const2 %s %s``
square: (a boolean)
    Take square of a volume.
    argument: ``-square``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
calc_mul: (a boolean or a float)
    Multiply N volumes or volume * constant.
    argument: ``-mult``
format_filetype: (a boolean)
    Use data type of first file (default).
    argument: ``-filetype``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
calc_or: (a boolean)
    Calculate  $\text{vol1} || \text{vol2} (|| \dots)$ .
    argument: ``-or``
test_ne: (a boolean or a float)
    Test for integer  $\text{vol1} != \text{vol2}$  or  $\text{vol1} != \text{const}$ .
    argument: ``-ne``
nisnan: (a boolean)
    Negation of -isnan.
    argument: ``-nisnan``
test_ge: (a boolean or a float)
    Test for  $\text{vol1} \geq \text{vol2}$  or  $\text{vol1} \geq \text{const}$ .

```

(continues on next page)

(continued from previous page)

```

    argument: ``-ge``
format_signed: (a boolean)
    Write signed integer data.
    argument: ``-signed``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
test_gt: (a boolean or a float)
    Test for vol1 > vol2 or vol1 > constant.
    argument: ``-gt``
format_int: (a boolean)
    Write out 32-bit integer data.
    argument: ``-int``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
output_illegal: (a boolean)
    Value to write out when an illegal operation is done. Default value:
    1.79769e+308
    argument: ``-illegal_value``
    mutually_exclusive: output_nan, output_zero, output_illegal_value
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``-clobber``
propagate_nan: (a boolean)
    Invalid data in any file at a voxel produces a NaN (default).
    argument: ``-propagate_nan``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
percentdiff: (a float)
    Percent difference between 2 volumes, thresholded (const def=0.0).
    argument: ``-percentdiff``
test_le: (a boolean or a float)
    Test for vol1 <= vol2 or vol1 <= const.
    argument: ``-le``
max_buffer_size_in_kb: (a long integer >= 0, nipy default value:
    4096)
    Specify the maximum size of the internal buffers (in kbytes).
    argument: ``-max_buffer_size_in_kb %d``
calc_not: (a boolean)
    Calculate !vol1.
    argument: ``-not``
sqrt: (a boolean)
    Take square root of a volume.
    argument: ``-sqrt``
format_long: (a boolean)
    Superseded by -int.
    argument: ``-long``
    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).
    argument: ``-unsigned``
    mutually_exclusive: format_filetype, format_byte, format_short,

```

(continues on next page)

(continued from previous page)

```

        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
test_lt: (a boolean or a float)
    Test for vol1 < vol2 or vol1 < constant.
    argument: ``-lt``
clamp: (a tuple of the form: (a float, a float))
    Clamp a volume to lie between two values.
    argument: ``-clamp -const2 %s %s``
copy_header: (a boolean)
    Copy all of the header from the first file (default for one file).
    argument: ``-copy_header``
    mutually_exclusive: copy_header, no_copy_header
format_float: (a boolean)
    Write out single-precision floating-point data.
    argument: ``-float``
    mutually_exclusive: format_filetype, format_byte, format_short,
        format_int, format_long, format_float, format_double,
        format_signed, format_unsigned
count_valid: (a boolean)
    Count the number of valid values in N volumes.
    argument: ``-count_valid``
nsegment: (a tuple of the form: (a float, a float))
    Opposite of -segment: within range = 0, outside range = 1.
    argument: ``-nsegment -const2 %s %s``
output_zero: (a boolean)
    Output zero when an illegal operation is done.
    argument: ``-zero``
    mutually_exclusive: output_nan, output_zero, output_illegal_value
isnan: (a boolean)
    Test for NaN values in vol1.
    argument: ``-isnan``
voxel_range: (a tuple of the form: (an integer (int or long), an
    integer (int or long)))
    Valid range for output data.
    argument: ``-range %d %d``
output_nan: (a boolean)
    Output NaN when an illegal operation is done (default).
    argument: ``-nan``
    mutually_exclusive: output_nan, output_zero, output_illegal_value
maximum: (a boolean)
    Find maximum of N volumes.
    argument: ``-maximum``
dimension: (a unicode string)
    Specify a dimension along which we wish to perform a calculation.
    argument: ``-dimension %s``

```

Outputs:

```

output_file: (an existing file name)
    output file

```

64.2.15 NlpFit[Link to code](#)Wraps the executable 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 nipyre.interfaces.minc import NlpFit
>>> from nipyre.interfaces.minc.testdata import nonempty_minc_data, nlp_config
>>> from nipyre.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]
source_mask: (an existing file name)
    Source mask to use during fitting.
    argument: ``-source_mask %s``
source: (an existing file name)
    source Minc file
    argument: ``%s``, position: -3
init_xfm: (an existing file name)
    Initial transformation (default identity).
    argument: ``-init_xfm %s``
target: (an existing file name)
    target Minc file
    argument: ``%s``, position: -2
config_file: (an existing file name)
    File containing the fitting configuration use.
    argument: ``-config_file %s``

[Optional]
input_grid_files: (a list of items which are a file name)
    input grid file(s)
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``-verbose``
output_xfm: (a file name)
    output xfm file
    argument: ``%s``, position: -1
clobber: (a boolean, nipyre default value: True)
    Overwrite existing file.
    argument: ``-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
```

Outputs:

```

output_grid: (an existing file name)
              output grid file
output_xfm: (an existing file name)
             output xfm file

```

64.2.16 Norm

[Link to code](#)

Wraps the executable command `mincnorm`.

Normalise a file between a max and minimum (possibly) using two histogram pct's.

Examples

```

>>> from nipy.interfaces.minc import Norm
>>> from nipy.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
            argument: ``%s``, position: -2

[Optional]
mask: (a file name)
      Calculate the image normalisation within a mask.
      argument: ``-mask %s``
upper: (a float)
      Upper real value to use.
      argument: ``-upper %s``
threshold_perc: (0.0 <= a floating point number <= 100.0)
                Threshold percentage (0.1 == lower 10% of intensity range) [default:
                0.1].
                argument: ``-threshold_perc %s``
output_file: (a file name)
             output file
             argument: ``%s``, position: -1
clobber: (a boolean, nipy default value: True)
         Overwrite existing file.
         argument: ``-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]
        argument: ``-cutoff %s``
lower: (a float)
       Lower real value to use.
       argument: ``-lower %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
threshold_bmt: (a boolean)
              Use the resulting image BiModalT as the threshold.

```

(continues on next page)

(continued from previous page)

```

        argument: ``-threshold_bmt``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
threshold: (a boolean)
    Threshold the image (set values below threshold_perc to -out_floor).
    argument: ``-threshold``
clamp: (a boolean, nipy default value: True)
    Force the ouput range between limits [default].
    argument: ``-clamp``
threshold_blur: (a float)
    Blur FWHM for intensity edges then thresholding [default: 2].
    argument: ``-threshold_blur %s``
out_ceil: (a float)
    Output files minimum [default: 100]
    argument: ``-out_ceil %s``
output_threshold_mask: (a file name)
    File in which to store the threshold mask.
    argument: ``-threshold_mask %s``
out_floor: (a float)
    Output files maximum [default: 0]
    argument: ``-out_floor %s``

```

Outputs:

```

output_threshold_mask: (a file name)
    threshold mask file
output_file: (an existing file name)
    output file

```

64.2.17 Pik[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -2

[Optional]
slice_y: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        Get a coronal (y) slice.
        argument: ``-y``
        mutually_exclusive: slice_z, slice_y, slice_x
auto_range: (a boolean)
        Automatically determine image range using a 5 and 95% PcT.
        (histogram)
        argument: ``--auto_range``
        mutually_exclusive: image_range, auto_range
width: (an integer (int or long))
        Autoscale the resulting image to have a fixed image width (in
        pixels).
        argument: ``--width %s``
image_range: (a tuple of the form: (a float, a float))
        Range of image values to use for pixel intensity.
        argument: ``--image_range %s %s``
        mutually_exclusive: image_range, auto_range
triplanar: (a boolean)
        Create a triplanar view of the input file.
        argument: ``--triplanar``
start: (an integer (int or long))
        Slice number to get. (note this is in voxel co-ordinates).
        argument: ``--slice %s``
clobber: (a boolean, nipy default value: True)
        Overwrite existing file.
        argument: ``-clobber``
depth: (8 or 16)
        Bitdepth for resulting image 8 or 16 (MSB machines only!)
        argument: ``--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
slice_z: (a boolean)
        Get an axial/transverse (z) slice.
        argument: ``-z``
        mutually_exclusive: slice_z, slice_y, slice_x
png: (a boolean)
        Output a png file (default).
        mutually_exclusive: jpg, png
tile_size: (an integer (int or long))
        Pixel size for each image in a triplanar.
        argument: ``--tilesize %s``
sagittal_offset_perc: (0 <= a long integer <= 100)
        Offset the sagittal slice by a percentage from the centre.
        argument: ``--sagittal_offset_perc %d``
title_size: (an integer (int or long))
        Font point size for the title.
        argument: ``--title_size %s``
        requires: title
annotated_bar: (a boolean)
        create an annotated bar to match the image (use height of the output
        image)
        argument: ``--anot_bar``
jpg: (a boolean)
        Output a jpg file.
        mutually_exclusive: jpg, png
title: (a boolean or a unicode string)

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
output_file: (a file name)
        output file
        argument: ``%s``, position: -1
minc_range: (a tuple of the form: (a float, a float))
        Valid range of values for MINC file.
        argument: ``--range %s %s``
scale: (an integer (int or long), nipy default value: 2)
        Scaling factor for resulting image. By default images are output at
        twice their original resolution.
        argument: ``--scale %s``
vertical_triplanar_view: (a boolean)
        Create a vertical triplanar view (Default).
        argument: ``--vertical``
        mutually_exclusive: vertical_triplanar_view,
        horizontal_triplanar_view
lookup: (a unicode string)
        Arguments to pass to minclookup
        argument: ``--lookup %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
slice_x: (a boolean)
        Get a sagittal (x) slice.
        argument: ``-x``
        mutually_exclusive: slice_z, slice_y, slice_x
horizontal_triplanar_view: (a boolean)
        Create a horizontal triplanar view.
        argument: ``--horizontal``
        mutually_exclusive: vertical_triplanar_view,
        horizontal_triplanar_view
sagittal_offset: (an integer (int or long))
        Offset the sagittal slice from the centre.
        argument: ``--sagittal_offset %s``

```

Outputs:

```

output_file: (an existing file name)
        output image

```

64.2.18 Resample[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -2

[Optional]
nearest_neighbour_interpolation: (a boolean)
    Do nearest neighbour interpolation.
    argument: ``-nearest_neighbour``
    mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
        nearest_neighbour_interpolation, sinc_interpolation
zstart: (a float)
    Start point along the Z dimension. Default value: 1.79769e+308.
    argument: ``-zstart %s``
    mutually_exclusive: start, start_x_y_or_z
    requires: xstart, ystart
format_byte: (a boolean)
    Write out byte data.
    argument: ``-byte``
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
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.
    argument: ``-start %s %s %s``
    mutually_exclusive: nelements, nelements_x_y_or_z
fill: (a boolean)
    Use a fill value for points outside of input volume.
    argument: ``-fill``
    mutually_exclusive: nofill, fill
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.
    argument: ``-dircos %s %s %s``
    mutually_exclusive: nelements, nelements_x_y_or_z
nokeep_real_range: (a boolean)
    Do not keep the real scale of the data (default).
    argument: ``-nokeep_real_range``
    mutually_exclusive: keep_real_range, nokeep_real_range
input_grid_files: (a list of items which are a file name)
    input grid file(s)
talairach: (a boolean)
    Output is in Talairach space.
    argument: ``-talairach``
ydircos: (a float)
    Direction cosines along the Y dimension. Default value: 1.79769e+308
    1.79769e+308 1.79769e+308.
    argument: ``-ydircos %s``
    mutually_exclusive: dircos, dircos_x_y_or_z
    requires: xdircos, zdircos
znelements: (an integer (int or long))
    Number of elements along the Z dimension.
    argument: ``-znelements %s``
    mutually_exclusive: nelements, nelements_x_y_or_z
    requires: xnelements, ynelements

```

(continues on next page)

(continued from previous page)

```

coronal_slices: (a boolean)
    Write out coronal slices
    argument: ``-coronal``
    mutually_exclusive: transverse, sagittal, coronal
like: (a file name)
    Specifies a model file for the resampling.
    argument: ``-like %s``
invert_transformation: (a boolean)
    Invert the transformation before using it.
    argument: ``-invert_transformation``
zstep: (an integer (int or long))
    Step size along the Z dimension. Default value: 0.
    argument: ``-zstep %s``
    mutually_exclusive: step, step_x_y_or_z
    requires: xstep, ystep
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).
    argument: ``-nelements %s %s %s``
    mutually_exclusive: nelements, nelements_x_y_or_z
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
transverse_slices: (a boolean)
    Write out transverse slices.
    argument: ``-transverse``
    mutually_exclusive: transverse, sagittal, coronal
vio_transform: (a boolean)
    VIO_Transform the input sampling with the transform (default).
    argument: ``-tfm_input_sampling``
    mutually_exclusive: vio_transform, no_input_sampling
ystep: (an integer (int or long))
    Step size along the Y dimension. Default value: 0.
    argument: ``-ystep %s``
    mutually_exclusive: step, step_x_y_or_z
    requires: xstep, zstep
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.
    argument: ``-width %s``
    requires: sinc_interpolation
keep_real_range: (a boolean)
    Keep the real scale of the input volume.
    argument: ``-keep_real_range``
    mutually_exclusive: keep_real_range, nokeep_real_range
format_int: (a boolean)
    Write out 32-bit integer data.
    argument: ``-int``
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
sinc_window_hanning: (a boolean)
    Set sinc window type to Hanning.
    argument: ``-hanning``
    mutually_exclusive: sinc_window_hanning, sinc_window_hamming
    requires: sinc_interpolation
output_range: (a tuple of the form: (a float, a float))

```

(continues on next page)

(continued from previous page)

```

Valid range for output data. Default value: -1.79769e+308
-1.79769e+308.
argument: ``-range %s %s``
xstep: (an integer (int or long))
Step size along the X dimension. Default value: 0.
argument: ``-xstep %s``
mutually_exclusive: step, step_x_y_or_z
requires: ystep, zstep
xstart: (a float)
Start point along the X dimension. Default value: 1.79769e+308.
argument: ``-xstart %s``
mutually_exclusive: start, start_x_y_or_z
requires: ystart, zstart
yelements: (an integer (int or long))
Number of elements along the Y dimension.
argument: ``-yelements %s``
mutually_exclusive: nelements, nelements_x_y_or_z
requires: xelements, zelements
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
format_short: (a boolean)
Write out short integer data.
argument: ``-short``
mutually_exclusive: format_byte, format_short, format_int,
format_long, format_float, format_double, format_signed,
format_unsigned
clobber: (a boolean, nipy default value: True)
Overwrite existing file.
argument: ``-clobber``
no_fill: (a boolean)
Use value zero for points outside of input volume.
argument: ``-nofill``
mutually_exclusive: nofill, fill
trilinear_interpolation: (a boolean)
Do trilinear interpolation.
argument: ``-trilinear``
mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
nearest_neighbour_interpolation, sinc_interpolation
sinc_window_hamming: (a boolean)
Set sinc window type to Hamming.
argument: ``-hamming``
mutually_exclusive: sinc_window_hanning, sinc_window_hamming
requires: sinc_interpolation
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipy default value: {})
Environment variables
two: (a boolean)
Create a MINC 2 output file.
argument: ``-2``
units: (a unicode string)
Specify the units of the output sampling.
argument: ``-units %s``
xdircos: (a float)
Direction cosines along the X dimension. Default value: 1.79769e+308
1.79769e+308 1.79769e+308.

```

(continues on next page)

(continued from previous page)

```

    argument: ``-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.
    argument: ``-xnelements %s``
    mutually_exclusive: nelements, nelements_x_y_or_z
    requires: ynelements, znelements
format_long: (a boolean)
    Superseded by -int.
    argument: ``-long``
    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).
    argument: ``-unsigned``
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
ystart: (a float)
    Start point along the Y dimension. Default value: 1.79769e+308.
    argument: ``-ystart %s``
    mutually_exclusive: start, start_x_y_or_z
    requires: xstart, zstart
format_signed: (a boolean)
    Write signed integer data.
    argument: ``-signed``
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
tricubic_interpolation: (a boolean)
    Do tricubic interpolation.
    argument: ``-tricubic``
    mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
        nearest_neighbour_interpolation, sinc_interpolation
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).
    argument: ``-step %s %s %s``
    mutually_exclusive: nelements, nelements_x_y_or_z
no_input_sampling: (a boolean)
    Use the input sampling without transforming (old behaviour).
    argument: ``-use_input_sampling``
    mutually_exclusive: vio_transform, no_input_sampling
format_float: (a boolean)
    Write out single-precision floating-point data.
    argument: ``-float``
    mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
fill_value: (a float)
    Specify a fill value for points outside of input volume. Default
    value: 1.79769e+308.
    argument: ``-fillvalue %s``
    requires: fill
standard_sampling: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        Set the sampling to standard values (step, start and dircos).
        argument: ``-standard_sampling``
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.
        argument: ``-origin %s %s %s``
spacetype: (a unicode string)
        Set the spacetype attribute to a specified string.
        argument: ``-spacetype %s``
sagittal_slices: (a boolean)
        Write out sagittal slices
        argument: ``-sagittal``
        mutually_exclusive: transverse, sagittal, coronal
zdircos: (a float)
        Direction cosines along the Z dimension.Default value: 1.79769e+308
        1.79769e+308 1.79769e+308.
        argument: ``-zdircos %s``
        mutually_exclusive: dircos, dircos_x_y_or_z
        requires: xdircos, ydircos
format_double: (a boolean)
        Write out double-precision floating-point data.
        argument: ``-double``
        mutually_exclusive: format_byte, format_short, format_int,
        format_long, format_float, format_double, format_signed,
        format_unsigned
sinc_interpolation: (a boolean)
        Do windowed sinc interpolation.
        argument: ``-sinc``
        mutually_exclusive: trilinear_interpolation, tricubic_interpolation,
        nearest_neighbour_interpolation, sinc_interpolation
transformation: (a file name)
        File giving world transformation. (Default = identity).
        argument: ``-transformation %s``

```

Outputs:

```

output_file: (an existing file name)
             output file

```

64.2.19 Reshape[Link to code](#)

Wraps the executable 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 nipy.interfaces.minc import Reshape
>>> from nipy.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
    argument: ``%s``, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``-verbose``
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``-clobber``
write_short: (a boolean)
    Convert to short integer data.
    argument: ``-short``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
output_file: (an existing file name)
    output file
```

64.2.20 ToEcat

[Link to code](#)

Wraps the executable command `minctoecat`.

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
    argument: ``%s``, position: -2
```

(continues on next page)

(continued from previous page)

```
[Optional]
voxels_as_integers: (a boolean)
    Voxel values are treated as integers, scale and calibration factors
    are set to unity
    argument: ``-label``
ignore_ecat_main: (a boolean)
    Ignore informations from the minc ecat-main variable.
    argument: ``-ignore_ecat_main``
no_decay_corr_fctr: (a boolean)
    Do not compute the decay correction factors
    argument: ``-no_decay_corr_fctr``
output_file: (a file name)
    output file
    argument: ``%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
ignore_study_variable: (a boolean)
    Ignore informations from the minc study variable.
    argument: ``-ignore_study_variable``
ignore_ecat_subheader_variable: (a boolean)
    Ignore informations from the minc ecat-subhdr variable.
    argument: ``-ignore_ecat_subheader_variable``
ignore_patient_variable: (a boolean)
    Ignore informations from the minc patient variable.
    argument: ``-ignore_patient_variable``
ignore_ecat_acquisition_variable: (a boolean)
    Ignore informations from the minc ecat_acquisition variable.
    argument: ``-ignore_ecat_acquisition_variable``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
ignore_acquisition_variable: (a boolean)
    Ignore informations from the minc acquisition variable.
    argument: ``-ignore_acquisition_variable``
```

Outputs:

```
output_file: (an existing file name)
    output file
```

64.2.21 ToRaw[Link to code](#)Wraps the executable command `minctoraw`.Dump a chunk of MINC file data. This program is largely superseded 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
    argument: ``%s``, position: -2

[Optional]
write_unsigned: (a boolean)
    Write out unsigned data.
    argument: ``-unsigned``
    mutually_exclusive: write_signed, write_unsigned
write_signed: (a boolean)
    Write out signed data.
    argument: ``-signed``
    mutually_exclusive: write_signed, write_unsigned
normalize: (a boolean)
    Normalize integer pixel values to file max and min.
    argument: ``-normalize``
    mutually_exclusive: normalize, nonnormalize
out_file: (a file name)
    argument: ``> %s``, position: -1
output_file: (a file name)
    output file
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.
    argument: ``-range %s %s``
write_int: (a boolean)
    Write out data as 32-bit integers.
    argument: ``-int``
    mutually_exclusive: write_byte, write_short, write_int, write_long,
        write_float, write_double
write_byte: (a boolean)
    Write out data as bytes.
    argument: ``-byte``
    mutually_exclusive: write_byte, write_short, write_int, write_long,
        write_float, write_double
nonnormalize: (a boolean)
    Turn off pixel normalization.
    argument: ``-nonnormalize``
    mutually_exclusive: normalize, nonnormalize
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
write_float: (a boolean)
    Write out data as single precision floating-point values.
    argument: ``-float``
    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.
    argument: ``-double``
    mutually_exclusive: write_byte, write_short, write_int, write_long,
```

(continues on next page)

(continued from previous page)

```

        write_float, write_double
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
write_long: (a boolean)
    Superseded by write_int.
    argument: ``-long``
    mutually_exclusive: write_byte, write_short, write_int, write_long,
        write_float, write_double
write_short: (a boolean)
    Write out data as short integers.
    argument: ``-short``
    mutually_exclusive: write_byte, write_short, write_int, write_long,
        write_float, write_double

```

Outputs:

```

output_file: (an existing file name)
    output file in raw format

```

64.2.22 VolSymm[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -3

[Optional]
fit_nonlinear: (a boolean)
    Fit using a non-linear xfm.
    argument: ``-nonlinear``
nofit: (a boolean)
    Use the input transformation instead of generating one.
    argument: ``-nofit``
x: (a boolean)
    Flip volume in x-plane (default).
    argument: ``-x``

```

(continues on next page)

(continued from previous page)

```

clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``-clobber``
output_file: (a file name)
    output file
    argument: ``%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
trans_file: (a file name)
    output xfm trans file
    argument: ``%s``, position: -2
input_grid_files: (a list of items which are a file name)
    input grid file(s)
y: (a boolean)
    Flip volume in y-plane.
    argument: ``-y``
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``-verbose``
fit_linear: (a boolean)
    Fit using a linear xfm.
    argument: ``-linear``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
config_file: (an existing file name)
    File containing the fitting configuration (nlpfit -help for info).
    argument: ``-config_file %s``
z: (a boolean)
    Flip volume in z-plane.
    argument: ``-z``

```

Outputs:

```

trans_file: (an existing file name)
    xfm trans file
output_grid: (an existing file name)
    output grid file
output_file: (an existing file name)
    output file

```

64.2.23 Volcentre[Link to code](#)

Wraps the executable 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
            argument: ``%s``, position: -2

[Optional]
com: (a boolean)
     Use the CoM of the volume for the new centre (via mincstats).
     Default: False
     argument: ``-com``
args: (a unicode string)
     Additional parameters to the command
     argument: ``%s``
verbose: (a boolean)
     Print out log messages. Default: False.
     argument: ``-verbose``
output_file: (a file name)
             output file
             argument: ``%s``, position: -1
zero_diracos: (a boolean)
              Set the direction cosines to identity [default].
              argument: ``-zero_diracos``
clobber: (a boolean, nipy default value: True)
          Overwrite existing file.
          argument: ``-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
centre: (a tuple of the form: (a float, a float, a float))
        Centre to use (x,y,z) [default: 0 0 0].
        argument: ``-centre %s %s %s``

```

Outputs:

```

output_file: (an existing file name)
             output file

```

64.2.24 Voliso[Link to code](#)

Wraps the executable command voliso.

Changes the steps and starts in order that the output volume has isotropic sampling.

Examples

```

>>> from nipy.interfaces.minc import Voliso
>>> from nipy.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
            argument: ``%s``, position: -2

```

(continues on next page)

(continued from previous page)

```
[Optional]
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``--verbose``
maxstep: (a float)
    The target maximum step desired in the output volume.
    argument: ``--maxstep %s``
minstep: (a float)
    The target minimum step desired in the output volume.
    argument: ``--minstep %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``--clobber``
avgstep: (a boolean)
    Calculate the maximum step from the average steps of the input
    volume.
    argument: ``--avgstep``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
output_file: (an existing file name)
    output file
```

64.2.25 Volpad

[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -2

[Optional]
clobber: (a boolean, nipy default value: True)
```

(continues on next page)

(continued from previous page)

```

        Overwrite existing file.
        argument: ``-clobber``
auto_freq: (a float)
    Frequency of voxels over bimodal threshold to stop at [default:
    500].
    argument: ``-auto_freq %s``
distance: (an integer (int or long))
    Padding distance (in voxels) [default: 4].
    argument: ``-distance %s``
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``-verbose``
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
auto: (a boolean)
    Automatically determine padding distances (uses -distance as max).
    Default: False.
    argument: ``-auto``
smooth_distance: (an integer (int or long))
    Smoothing distance (in voxels) [default: 4].
    argument: ``-smooth_distance %s``
smooth: (a boolean)
    Smooth (blur) edges before padding. Default: False.
    argument: ``-smooth``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

output_file: (an existing file name)
    output file

```

64.2.26 XfmAvg[Link to code](#)

Wraps the executable 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

```

(continues on next page)

(continued from previous page)

```
>>> xfmavg = XfmAvg(input_files=[xfm1, xfm2])
>>> xfmavg.run()
```

Inputs:

```
[Mandatory]
input_files: (a list of items which are a file name)
    input file(s)
    argument: ``%s``, position: -2

[Optional]
ignore_linear: (a boolean)
    opposite of -avg_linear.
    argument: ``-ignore_linear``
input_grid_files: (a list of items which are a file name)
    input grid file(s)
avg_nonlinear: (a boolean)
    average the non-linear part [default].
    argument: ``-avg_nonlinear``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``-verbose``
clobber: (a boolean, nipy default value: True)
    Overwrite existing file.
    argument: ``-clobber``
avg_linear: (a boolean)
    average the linear part [default].
    argument: ``-avg_linear``
output_file: (a file name)
    output file
    argument: ``%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
ignore_nonlinear: (a boolean)
    opposite of -avg_nonlinear.
    argument: ``-ignore_nonlinear``
```

Outputs:

```
output_grid: (an existing file name)
    output grid file
output_file: (an existing file name)
    output file
```

64.2.27 XfmConcat[Link to code](#)Wraps the executable command `xfmconcat`.Concatenate transforms together. The output transformation is equivalent to applying `input1.xfm`, then `input2.xfm`, ..., in that order.

Examples

```
>>> from nipyre.interfaces.minc import XfmConcat
>>> from nipyre.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)
    argument: ``%s``, position: -2

[Optional]
input_grid_files: (a list of items which are a file name)
    input grid file(s)
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``-verbose``
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
clobber: (a boolean, nipyre default value: True)
    Overwrite existing file.
    argument: ``-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
```

Outputs:

```
output_grids: (a list of items which are an existing file name)
    output grids
output_file: (an existing file name)
    output file
```

64.2.28 XfmInvert

[Link to code](#)

Wraps the executable command `xfminvert`.

Invert an xfm transform file.

Examples

```
>>> from nipyre.interfaces.minc import XfmAvg
>>> from nipyre.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
    argument: ``%s``, position: -2

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    Print out log messages. Default: False.
    argument: ``-verbose``
output_file: (a file name)
    output file
    argument: ``%s``, position: -1
clobber: (a boolean, nipyre default value: True)
    Overwrite existing file.
    argument: ``-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
```

Outputs:

```
output_grid: (an existing file name)
    output grid file
output_file: (an existing file name)
    output file
```

64.3 interfaces.minc.testdata

64.3.1 nonempty_minc_data()

[Link to code](#)

65.1 interfaces.mipav.developer

65.1.1 JistBrainMgdmSegmentation

[Link to code](#)

Wraps the executable 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:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inAdjust: ('true' or 'false')
    Adjust intensity priors
    argument: ``--inAdjust %s``
outPosterior2: (a boolean or a file name)
    Posterior Maximum Memberships (4D)
    argument: ``--outPosterior2 %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
environ: (a dictionary with keys which are a 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
inCompute: ('true' or 'false')
    Compute posteriors
    argument: ``--inCompute %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
```

(continues on next page)

(continued from previous page)

```

inFLAIR: (an existing file name)
    FLAIR Image
    argument: ``--inFLAIR %s``
inMax: (an integer (int or long))
    Max iterations
    argument: ``--inMax %d``
outSegmented: (a boolean or a file name)
    Segmented Brain Image
    argument: ``--outSegmented %s``
inMP2RAGE2: (an existing file name)
    MP2RAGE T1-weighted Image
    argument: ``--inMP2RAGE2 %s``
outPosterior3: (a boolean or a file name)
    Posterior Maximum Labels (4D)
    argument: ``--outPosterior3 %s``
inTopology: ('26/6' or '6/26' or '18/6' or '6/18' or '6/6' or 'wcs'
    or 'wco' or 'no')
    Topology
    argument: ``--inTopology %s``
inData: (a float)
    Data weight
    argument: ``--inData %f``
inCurvature: (a float)
    Curvature weight
    argument: ``--inCurvature %f``
inOutput: ('segmentation' or 'memberships')
    Output images
    argument: ``--inOutput %s``
inPV: (an existing file name)
    PV / Dura Image
    argument: ``--inPV %s``
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
inAtlas: (an existing file name)
    Atlas file
    argument: ``--inAtlas %s``
inPosterior: (a float)
    Posterior scale (mm)
    argument: ``--inPosterior %f``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
inMin: (a float)
    Min change
    argument: ``--inMin %f``
outLevelset: (a boolean or a file name)
    Levelset Boundary Image
    argument: ``--outLevelset %s``
inMPRAGE: (an existing file name)
    MPRAGE T1-weighted Image
    argument: ``--inMPRAGE %s``
inMP2RAGE: (an existing file name)
    MP2RAGE T1 Map Image
    argument: ``--inMP2RAGE %s``
inSteps: (an integer (int or long))
    Steps

```

(continues on next page)

(continued from previous page)

```
argument: ``--inSteps %d``
```

Outputs:

```
outLevelset: (an existing file name)
    Levelset Boundary Image
outPosterior3: (an existing file name)
    Posterior Maximum Labels (4D)
outPosterior2: (an existing file name)
    Posterior Maximum Memberships (4D)
outSegmented: (an existing file name)
    Segmented Brain Image
```

65.1.2 JistBrainMp2rageDuraEstimation[Link to code](#)

Wraps the executable 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:

```
[Optional]
inDistance: (a float)
    Distance to background (mm)
    argument: ``--inDistance %f``
outDura: (a boolean or a file name)
    Dura Image
    argument: ``--outDura %s``
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
inSkull: (an existing file name)
    Skull Stripping Mask
    argument: ``--inSkull %s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
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.
    argument: ``--inoutput %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
```

(continues on next page)

(continued from previous page)

```

        value of class 'str', nipy default value: {})
    Environment variables
inSecond: (an existing file name)
    Second inversion (Inv2) Image
    argument: ``--inSecond %s``

```

Outputs:

```

outDura: (an existing file name)
    Dura Image

```

65.1.3 JistBrainMp2rageSkullStripping[Link to code](#)

Wraps the executable command “`java edu.jhu.ece.iac1.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:

```

[Optional]
inFilter: (an existing file name)
    Filter Image (opt)
    argument: ``--inFilter %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
outMasked3: (a boolean or a file name)
    Masked Filter Image
    argument: ``--outMasked3 %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inTlweighted: (an existing file name)
    T1-weighted (UNI) Image (opt)
    argument: ``--inTlweighted %s``
inTl: (an existing file name)
    T1 Map (T1_Images) Image (opt)
    argument: ``--inTl %s``
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %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
outMasked2: (a boolean or a file name)
    Masked T1-weighted Image
    argument: ``--outMasked2 %s``
inSkip: ('true' or 'false')
    Skip zero values
    argument: ``--inSkip %s``
outMasked: (a boolean or a file name)

```

(continues on next page)

(continued from previous page)

```

        Masked T1 Map Image
        argument: ``--outMasked %s``
xPrefExt: ('nrrd')
        Output File Type
        argument: ``--xPrefExt %s``
xDefaultMem: (an integer (int or long))
        Set default maximum heap size
        argument: ``-xDefaultMem %d``
outBrain: (a boolean or a file name)
        Brain Mask Image
        argument: ``--outBrain %s``
inSecond: (an existing file name)
        Second inversion (Inv2) Image
        argument: ``--inSecond %s``

```

Outputs:

```

outMasked: (an existing file name)
        Masked T1 Map Image
outMasked3: (an existing file name)
        Masked Filter Image
outBrain: (an existing file name)
        Brain Mask Image
outMasked2: (an existing file name)
        Masked T1-weighted Image

```

65.1.4 JistBrainPartialVolumeFilter[Link to code](#)

Wraps the executable 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:

```

[Optional]
xMaxProcess: (an integer (int or long), nipy default value: 1)
        Set default maximum number of processes.
        argument: ``-xMaxProcess %d``
inInput: (an existing file name)
        Input Image
        argument: ``--inInput %s``
outPartial: (a boolean or a file name)
        Partial Volume Image
        argument: ``--outPartial %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
xDefaultMem: (an integer (int or long))
        Set default maximum heap size
        argument: ``-xDefaultMem %d``
inoutput: ('probability' or 'intensity')
        output
        argument: ``--inoutput %s``

```

(continues on next page)

(continued from previous page)

```

inPV: ('bright' or 'dark' or 'both')
    Outputs the raw intensity values or a probability score for the
    partial volume regions.
    argument: ``--inPV %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

outPartial: (an existing file name)
    Partial Volume Image

```

65.1.5 JistCortexSurfaceMeshInflation[Link to code](#)

Wraps the executable command “`java edu.jhu.ece.iacI.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:**

```

[Optional]
outOriginal: (a boolean or a file name)
    Original Surface
    argument: ``--outOriginal %s``
inTopology: ('26/6' or '6/26' or '18/6' or '6/18' or '6/6' or 'wcs'
    or 'wco' or 'no')
    Topology
    argument: ``--inTopology %s``
inLevelset: (an existing file name)
    Levelset Image
    argument: ``--inLevelset %s``
inLorentzian: ('true' or 'false')
    Lorentzian Norm
    argument: ``--inLorentzian %s``
inStep: (an integer (int or long))
    Step Size
    argument: ``--inStep %d``
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``--xMaxProcess %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

```

(continues on next page)

(continued from previous page)

```

        value of class 'str', nipy default value: {})
    Environment variables
inSOR: (a float)
    SOR Parameter
    argument: ``--inSOR %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inMean: (a float)
    Mean Curvature Threshold
    argument: ``--inMean %f``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
inMax: (an integer (int or long))
    Max Iterations
    argument: ``--inMax %d``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
outInflated: (a boolean or a file name)
    Inflated Surface
    argument: ``--outInflated %s``

```

Outputs:

```

outOriginal: (an existing file name)
    Original Surface
outInflated: (an existing file name)
    Inflated Surface

```

65.1.6 JistIntensityMp2rageMasking[Link to code](#)

Wraps the executable command “`java edu.jhu.ece.iacl.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:

```

[Optional]
outSignal: (a boolean or a file name)
    Signal Proba Image
    argument: ``--outSignal_Proba %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inTlweighted: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

T1-weighted (UNI) Image
argument: ``--inTlweighted %s``
xMaxProcess: (an integer (int or long), nipy default value: 1)
Set default maximum number of processes.
argument: ``-xMaxProcess %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
outSignal2: (a boolean or a file name)
Signal Mask Image
argument: ``--outSignal_Mask %s``
outMasked2: (a boolean or a file name)
Masked Iso Image
argument: ``--outMasked_Tlweighted %s``
inMasking: ('binary' or 'proba')
Whether to use a binary threshold or a weighted average based on the
probability.
argument: ``--inMasking %s``
inBackground: ('exponential' or 'half-normal')
Model distribution for background noise (default is half-normal,
exponential is more stringent).
argument: ``--inBackground %s``
inSkip: ('true' or 'false')
Skip zero values
argument: ``--inSkip %s``
outMasked: (a boolean or a file name)
Masked T1 Map Image
argument: ``--outMasked_T1_Map %s``
xPrefExt: ('nrrd')
Output File Type
argument: ``--xPrefExt %s``
xDefaultMem: (an integer (int or long))
Set default maximum heap size
argument: ``-xDefaultMem %d``
inQuantitative: (an existing file name)
Quantitative T1 Map (T1_Images) Image
argument: ``--inQuantitative %s``
inSecond: (an existing file name)
Second inversion (Inv2) Image
argument: ``--inSecond %s``

```

Outputs:

```

outSignal: (an existing file name)
Signal Proba Image
outMasked: (an existing file name)
Masked T1 Map Image
outSignal2: (an existing file name)
Signal Mask Image
outMasked2: (an existing file name)
Masked Iso Image

```

65.1.7 JistLaminarProfileCalculator[Link to code](#)Wraps the executable command `java edu.jhu.ece.iacl.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:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
inMask: (an existing file name)
    Mask Image (opt, 3D or 4D)
    argument: ``--inMask %s``
outResult: (a boolean or a file name)
    Result
    argument: ``--outResult %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
xMaxProcess: (an integer (int or long), nipyte default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
inIntensity: (an existing file name)
    Intensity Profile Image
    argument: ``--inIntensity %s``
environ: (a dictionary with keys which are a 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
incomputed: ('mean' or 'stdev' or 'skewness' or 'kurtosis')
    computed statistic
    argument: ``--incomputed %s``
```

Outputs:

```
outResult: (an existing file name)
    Result
```

65.1.8 JistLaminarProfileGeometry

[Link to code](#)

Wraps the executable command “java edu.jhu.ece.iacI.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:

```

[Optional]
insmoothing: (a float)
    smoothing parameter
    argument: ``--insmoothing %f``
inoutside: (a float)
    outside extension (mm)
    argument: ``--inoutside %f``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
inProfile: (an existing file name)
    Profile Surface Image
    argument: ``--inProfile %s``
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %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
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outResult: (a boolean or a file name)
    Result
    argument: ``--outResult %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
inregularization: ('none' or 'Gaussian')
    regularization
    argument: ``--inregularization %s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %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
    argument: ``--incomputed %s``

```

Outputs:

```

outResult: (an existing file name)
    Result

```

65.1.9 JistLaminarProfileSampling[Link to code](#)

Wraps the executable command “`java edu.jhu.ece.iacI.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:


```
[Optional]
inCortex: (an existing file name)
    Cortex Mask (opt)
    argument: ``--inCortex %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
xMaxProcess: (an integer (int or long), nipyre default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
outProfilemapped: (a boolean or a file name)
    Profile-mapped Intensity Image
    argument: ``--outProfilemapped %s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
outProfile2: (a boolean or a file name)
    Profile 4D Mask
    argument: ``--outProfile2 %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
inProfile: (an existing file name)
    Profile Surface Image
    argument: ``--inProfile %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
inIntensity: (an existing file name)
    Intensity Image
    argument: ``--inIntensity %s``
environ: (a dictionary with keys which are a 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
```

Outputs:

```
outProfilemapped: (an existing file name)
    Profile-mapped Intensity Image
outProfile2: (an existing file name)
    Profile 4D Mask
```

65.1.10 JistLaminarROIAveraging[Link to code](#)

Wraps the executable command “`java edu.jhu.ece.iacI.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:**

```
[Optional]
inROI2: (a unicode string)
    ROI Name
```

(continues on next page)

(continued from previous page)

```

        argument: ``--inROI2 %s``
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
inMask: (an existing file name)
    Mask Image (opt, 3D or 4D)
    argument: ``--inMask %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inROI: (an existing file name)
    ROI Mask
    argument: ``--inROI %s``
inIntensity: (an existing file name)
    Intensity Profile Image
    argument: ``--inIntensity %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outROI3: (a boolean or a file name)
    ROI Average
    argument: ``--outROI3 %s``

```

Outputs:

```

outROI3: (an existing file name)
    ROI Average

```

65.1.11 JistLaminarVolumetricLayering[Link to code](#)

Wraps the executable 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. Waehnert MD, Dinse J, Weiss M, Streicher MN, Waehnert P, Geyer S, Turner R, Bazin PL, Anatomically motivated modeling of cortical laminae, Neuroimage, 2013.

version: 3.0.RC

contributor: Miriam Waehnert (waehnert@cbs.mpg.de) <http://www.cbs.mpg.de/>

Inputs:

```

[Optional]
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``

```

(continues on next page)

(continued from previous page)

```

inNumber: (an integer (int or long))
    Number of layers
    argument: ``--inNumber %d``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
outDiscrete: (a boolean or a file name)
    Discrete sampled layers
    argument: ``--outDiscrete %s``
inLayering2: ('outward' or 'inward')
    Layering direction
    argument: ``--inLayering2 %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
inInner: (an existing file name)
    Inner Distance Image (GM/WM boundary)
    argument: ``--inInner %s``
environ: (a dictionary with keys which are a 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
inLayering: ('distance-preserving' or 'volume-preserving')
    Layering method
    argument: ``--inLayering %s``
inpresmooth: ('true' or 'false')
    pre-smooth cortical surfaces
    argument: ``--inpresmooth %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inratio: (a float)
    ratio smoothing kernel size (voxels)
    argument: ``--inratio %f``
outContinuous: (a boolean or a file name)
    Continuous depth measurement
    argument: ``--outContinuous %s``
inOuter: (an existing file name)
    Outer Distance Image (CSF/GM boundary)
    argument: ``--inOuter %s``
inMin: (a float)
    Min change ratio for narrow band evolution
    argument: ``--inMin %f``
inMax: (an integer (int or long))
    Max iterations for narrow band evolution
    argument: ``--inMax %d``
outLayer: (a boolean or a file name)
    Layer boundary surfaces
    argument: ``--outLayer %s``
incurvature: (an integer (int or long))
    curvature approximation scale (voxels)
    argument: ``--incurvature %d``
inTopology: ('26/6' or '6/26' or '18/6' or '6/18' or '6/6' or 'wcs'
    or 'wco' or 'no')

```

(continues on next page)

(continued from previous page)

```
Topology
argument: ``--inTopology %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
```

65.1.12 MedicAlgorithmImageCalculator[Link to code](#)

Wraps the executable command `“java edu.jhu.ece.iac1.jist.cli.run edu.jhu.ece.iac1.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.iac1.ece.jhu.edu/>

Inputs:

```
[Optional]
inVolume: (an existing file name)
    Volume 1
    argument: ``--inVolume %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
inVolume2: (an existing file name)
    Volume 2
    argument: ``--inVolume2 %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
outResult: (a boolean or a file name)
    Result Volume
    argument: ``--outResult %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
inOperation: ('Add' or 'Subtract' or 'Multiply' or 'Divide' or 'Min'
    or 'Max')
    Operation
    argument: ``--inOperation %s``
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %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
```

(continues on next page)

(continued from previous page)

```
value of class 'str', nipy default value: {}
Environment variables
```

Outputs:

```
outResult: (an existing file name)
Result Volume
```

65.1.13 MedicAlgorithmLesionToads[Link to code](#)

Wraps the executable 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:

```
[Optional]
outWM: (a boolean or a file name)
    WM Mask
    argument: ``--outWM %s``
inSmoothing: (a float)
    Controls the effect of neighborhood voxels on the membership
    argument: ``--inSmoothing %f``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
outMembership: (a boolean or a file name)
    Membership Functions
    argument: ``--outMembership %s``
inAtlas5: (a float)
    Controls the effect of the statistical atlas on the segmentation
    argument: ``--inAtlas5 %f``
inInclude: ('true' or 'false')
    Include lesion in WM class in hard classification
    argument: ``--inInclude %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
inAtlas6: ('rigid' or 'multi_fully_affine')
    Atlas alignment
    argument: ``--inAtlas6 %s``
inConnectivity: ('(26,6)' or '(6,26)' or '(6,18)' or '(18,6)')
    Connectivity (foreground,background)
    argument: ``--inConnectivity %s``
outSulcal: (a boolean or a file name)
```

(continues on next page)

(continued from previous page)

```

        Sulcal CSF Membership
        argument: ``--outSulcal %s``
outFilled: (a boolean or a file name)
        Filled WM Membership
        argument: ``--outFilled %s``
inOutput3: ('true' or 'false')
        Output the estimated inhomogeneity field
        argument: ``--inOutput3 %s``
inFLAIR: (an existing file name)
        FLAIR Image
        argument: ``--inFLAIR %s``
inAtlas2: (an existing file name)
        Atlas File - With Lesions
        argument: ``--inAtlas2 %s``
inT1_SPGR: (an existing file name)
        T1_SPGR Image
        argument: ``--inT1_SPGR %s``
outHard: (a boolean or a file name)
        Hard segmentation
        argument: ``--outHard %s``
inMaximum5: (an integer (int or long))
        Maximum iterations
        argument: ``--inMaximum5 %d``
inMaximum: (an integer (int or long))
        Maximum distance from the interventricular WM boundary to downweight
        the lesion membership to avoid false positives
        argument: ``--inMaximum %d``
outInhomogeneity: (a boolean or a file name)
        Inhomogeneity Field
        argument: ``--outInhomogeneity %s``
inMaximum3: (an integer (int or long))
        Maximum InterVentricular Distance
        argument: ``--inMaximum3 %d``
outLesion: (a boolean or a file name)
        Lesion Segmentation
        argument: ``--outLesion %s``
xMaxProcess: (an integer (int or long), nipy default value: 1)
        Set default maximum number of processes.
        argument: ``-xMaxProcess %d``
inCorrect: ('true' or 'false')
        Correct MR field inhomogeneity.
        argument: ``--inCorrect %s``
inOutput: ('hard segmentation' or 'hard segmentation+memberships' or
        'cruise inputs' or 'dura removal inputs')
        Output images
        argument: ``--inOutput %s``
inAtlas3: (an existing file name)
        Atlas File - No Lesion - T1 and FLAIR
        argument: ``--inAtlas3 %s``
inMaximum2: (an integer (int or long))
        Maximum Ventricle Distance
        argument: ``--inMaximum2 %d``
inAtlas: ('With Lesion' or 'No Lesion')
        Atlas to Use
        argument: ``--inAtlas %s``
inOutput2: ('true' or 'false')
        Output the hard classification using maximum membership (not

```

(continues on next page)

(continued from previous page)

```

        neceesarily topologically correct)
        argument: ``--inOutput2 %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
outHard2: (a boolean or a file name)
        Hard segmentationfrom memberships
        argument: ``--outHard2 %s``
xDefaultMem: (an integer (int or long))
        Set default maximum heap size
        argument: ``-xDefaultMem %d``
inMaximum4: (a float)
        Maximum amount of relative change in the energy function considered
        as the convergence criteria
        argument: ``--inMaximum4 %f``
outCortical: (a boolean or a file name)
        Cortical GM Membership
        argument: ``--outCortical %s``
inT1_MPRAGE: (an existing file name)
        T1_MPRAGE Image
        argument: ``--inT1_MPRAGE %s``
inAtlas4: (an existing file name)
        Atlas File - No Lesion - T1 Only
        argument: ``--inAtlas4 %s``

```

Outputs:

```

outHard: (an existing file name)
        Hard segmentation
outWM: (an existing file name)
        WM Mask
outInhomogeneity: (an existing file name)
        Inhomogeneity Field
outHard2: (an existing file name)
        Hard segmentationfrom memberships
outMembership: (an existing file name)
        Membership Functions
outLesion: (an existing file name)
        Lesion Segmentation
outCortical: (an existing file name)
        Cortical GM Membership
outSulcal: (an existing file name)
        Sulcal CSF Membership
outFilled: (an existing file name)
        Filled WM Membership

```

65.1.14 MedicAlgorithmMipavReorient[Link to code](#)

Wraps the executable command “`java edu.jhu.ece.iacl.jist.cli.run edu.jhu.ece.iacl.plugins.utilities.volume.MedicAlgorithmMipav`”.

title: Reorient Volume

category: Developer Tools

description: Reorient a volume to a particular anatomical orientation.

version: .alpha

Inputs:

```

[Optional]
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
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
    argument: ``--inInterpolation %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)
    argument: ``--inUser %s``
inTemplate: (an existing file name)
    Template
    argument: ``--inTemplate %s``
inResolution: ('Unchanged' or 'Finest cubic' or 'Coarsest cubic' or
    'Same as template')
    Resolution
    argument: ``--inResolution %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %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)
    argument: ``--inUser2 %s``
inUser4: ('Axial' or 'Coronal' or 'Sagittal' or 'Unknown')
    User defined Image Orientation
    argument: ``--inUser4 %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outReoriented: (a list of items which are a file name)
    Reoriented Volume
    argument: ``--outReoriented %s``
inNew: ('Dicom axial' or 'Dicom coronal' or 'Dicom sagittal' or 'User
    defined')
    New image orientation
    argument: ``--inNew %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)
    argument: ``--inUser3 %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``

```

(continues on next page)

(continued from previous page)

```
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
inSource: (a list of items which are a file name)
    Source
    argument: ``--inSource %s``
```

Outputs:

None

65.1.15 MedicAlgorithmN3

[Link to code](#)

Wraps the executable command “`java edu.jhu.ece.iac1.jist.cli.run edu.jhu.ece.iac1.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:

```
[Optional]
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.
    argument: ``--inAutomatic %s``
xMaxProcess: (an integer (int or long), nipyre default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
inMaximum: (an integer (int or long))
    Maximum number of Iterations
    argument: ``--inMaximum %d``
outInhomogeneity: (a boolean or a file name)
    Inhomogeneity Corrected Volume
    argument: ``--outInhomogeneity %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--inEnd %f``
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.
    argument: ``--inKernel %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
outInhomogeneity2: (a boolean or a file name)
    Inhomogeneity Field
    argument: ``--outInhomogeneity2 %s``
```

(continues on next page)

(continued from previous page)

```

xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
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.
    argument: ``--inSubsample %f``
inInput: (an existing file name)
    Input Volume
    argument: ``--inInput %s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
inWeiner: (a float)
    Usually between 0.0-1.0
    argument: ``--inWeiner %f``
inSignal: (a float)
    Default = min + 1, Values at less than threshold are treated as part
    of the background
    argument: ``--inSignal %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.
    argument: ``--inField %f``

```

Outputs:

```

outInhomogeneity2: (an existing file name)
    Inhomogeneity Field
outInhomogeneity: (an existing file name)
    Inhomogeneity Corrected Volume

```

65.1.16 MedicAlgorithmSPECTRE2010[Link to code](#)

Wraps the executable command “`java edu.jhu.ece.iacI.jist.cli.run edu.jhu.ece.iacI.plugins.segmentation.skull_strip.MedicAlgorithm`”.

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:

```

[Optional]
outSplitHalves: (a boolean or a file name)
    Skullstripped mask of the brain with the hemispheres divided.
    argument: ``--outSplitHalves %s``
inNumber: (an integer (int or long))
    Number of iterations
    argument: ``--inNumber %d``
inFine: (a float)
    Fine angle increment
    argument: ``--inFine %f``
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.
    argument: ``--outOriginal %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
    argument: ``--inRegistration %s``
inMultithreading: ('true' or 'false')
    Set to false by default, this parameter controls the multithreaded
    behavior of the linear registration.
    argument: ``--inMultithreading %s``
outPrior: (a boolean or a file name)
    Probability prior from the atlas registrations
    argument: ``--outPrior %s``
inSkip: ('true' or 'false')
    Skip multilevel search (Assume images are close to alignment)
    argument: ``--inSkip %s``
inInitial2: (a float)
    Initial probability threshold
    argument: ``--inInitial2 %f``
inOutput3: ('true' or 'false')
    Output Split-Halves?
    argument: ``--inOutput3 %s``
inMultiple: (an integer (int or long))
    Multiple of tolerance to bracket the minimum
    argument: ``--inMultiple %d``
outMidsagittal: (a boolean or a file name)
    Plane dividing the brain hemispheres
    argument: ``--outMidsagittal %s``
inUse: ('true' or 'false')
    Use the max of the min resolutions of the two datasets when
    resampling
    argument: ``--inUse %s``
inDegrees: ('Rigid - 6' or 'Global rescale - 7' or 'Specific rescale
    - 9' or 'Affine - 12')
    Degrees of freedom
    argument: ``--inDegrees %s``
inInput: (an existing file name)
    Input volume to be skullstripped.
    argument: ``--inInput %s``
inSmoothing: (a float)
    argument: ``--inSmoothing %f``
inBackground: (a float)

```

(continues on next page)

(continued from previous page)

```

        argument: ``--inBackground %f``
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
inFind: ('true' or 'false')
    Find Midsagittal Plane
    argument: ``--inFind %s``
inSubsample: ('true' or 'false')
    Subsample image for speed
    argument: ``--inSubsample %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
    argument: ``--inOutput5 %s``
inOutput2: ('true' or 'false')
    Output Plane?
    argument: ``--inOutput2 %s``
inRun: ('true' or 'false')
    Run Smooth Brain Mask
    argument: ``--inRun %s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``-xPrefExt %s``
inOutput4: ('true' or 'false')
    Output Segmentation on Plane?
    argument: ``--inOutput4 %s``
inMMC: (an integer (int or long))
    The size of the dilation step within the Modified Morphological
    Closing.
    argument: ``--inMMC %d``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
outd0: (a boolean or a file name)
    Initial Brainmask
    argument: ``--outd0 %s``
outFANTASM: (a boolean or a file name)
    Tissue classification of of the whole input volume.
    argument: ``--outFANTASM %s``
outStripped: (a boolean or a file name)
    Skullstripped result of the input volume with just the brain.
    argument: ``--outStripped %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
inNumber2: (an integer (int or long))
    Number of minima from Level 8 to test at Level 4
    argument: ``--inNumber2 %d``
inMinimum2: (a float)
    Minimum angle
    argument: ``--inMinimum2 %f``
outSegmentation: (a boolean or a file name)
    2D image showing the tissue classification on the midsagittal plane
    argument: ``--outSegmentation %s``
inMaximum: (a float)

```

(continues on next page)

(continued from previous page)

```

Maximum angle
argument: ``--inMaximum %f``
inMMC2: (an integer (int or long))
The size of the erosion step within the Modified Morphological
Closing.
argument: ``--inMMC2 %d``
inResample: ('true' or 'false')
Determines if the data is resampled to be isotropic during the
processing.
argument: ``--inResample %s``
inApply: ('All' or 'X' or 'Y' or 'Z')
Apply rotation
argument: ``--inApply %s``
inOutput: ('true' or 'false')
Determines if the output results are transformed back into the space
of the original input image.
argument: ``--inOutput %s``
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.
argument: ``--inImage %s``
outMask: (a boolean or a file name)
Binary Mask of the skullstripped result with just the brain
argument: ``--outMask %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.
argument: ``--inInhomogeneity %s``
inAtlas: (an existing file name)
SPECTRE atlas description file. A text file enumerating atlas files
and landmarks.
argument: ``--inAtlas %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.
argument: ``--inInitial %d``
inCoarse: (a float)
Coarse angle increment
argument: ``--inCoarse %f``
xDefaultMem: (an integer (int or long))
Set default maximum heap size
argument: ``-xDefaultMem %d``
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
inMinimum: (a float)
Minimum probability threshold
argument: ``--inMinimum %f``
inCost: ('Correlation ratio' or 'Least squares' or 'Normalized cross
correlation' or 'Normalized mutual information')
Cost function
argument: ``--inCost %s``

```

Outputs:

```

outSegmentation: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    2D image showing the tissue classification on the midsagittal plane
outPrior: (an existing file name)
    Probability prior from the atlas registrations
outSplitHalves: (an existing file name)
    Skullstripped mask of the brain with the hemispheres divided.
outMidsagittal: (an existing file name)
    Plane dividing the brain hemispheres
outd0: (an existing file name)
    Initial Brainmask
outFANTASM: (an existing file name)
    Tissue classification of of the whole input volume.
outStripped: (an existing file name)
    Skullstripped result of the input volume with just the brain.
outOriginal: (an existing file name)
    If Output in Original Space Flag is true then outputs the original
    input volume. Otherwise outputs the axialy reoriented input volume.
outMask: (an existing file name)
    Binary Mask of the skullstripped result with just the brain

```

65.1.17 MedicAlgorithmThresholdToBinaryMask

[Link to code](#)

Wraps the executable command `“java edu.jhu.ece.iac1.jist.cli.run edu.jhu.ece.iac1.plugins.utilities.volume.MedicAlgorithmThresh`

`“.`
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.iac1.ece.jhu.edu/>

Inputs:

```

[Optional]
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
inMaximum: (a float)
    Maximum threshold value.
    argument: ``--inMaximum %f``
inLabel: (a list of items which are a file name)
    Input volumes
    argument: ``--inLabel %s``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
inMinimum: (a float)
    Minimum threshold value.

```

(continues on next page)

(continued from previous page)

```

    argument: ``--inMinimum %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
outBinary: (a list of items which are a file name)
    Binary Mask
    argument: ``--outBinary %s``
inUse: ('true' or 'false')
    Use the images max intensity as the max value of the range.
    argument: ``--inUse %s``

```

Outputs:

None

65.1.18 RandomVol

[Link to code](#)

Wraps the executable command “`java edu.jhu.ece.iacjlist.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/jlist/>

Inputs:

```

[Optional]
xMaxProcess: (an integer (int or long), nipy default value: 1)
    Set default maximum number of processes.
    argument: ``-xMaxProcess %d``
inMaximum: (an integer (int or long))
    Maximum Value
    argument: ``--inMaximum %d``
outRand1: (a boolean or a file name)
    Rand1
    argument: ``--outRand1 %s``
inSize: (an integer (int or long))
    Size of Volume in X direction
    argument: ``--inSize %d``
inSize2: (an integer (int or long))
    Size of Volume in Y direction
    argument: ``--inSize2 %d``
inStandard: (an integer (int or long))
    Standard Deviation for Normal Distribution
    argument: ``--inStandard %d``
null: (a unicode string)
    Execution Time
    argument: ``--null %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
inSize3: (an integer (int or long))
    Size of Volume in Z direction
    argument: ``--inSize3 %d``

```

(continues on next page)

(continued from previous page)

```
inLambda: (a float)
    Lambda Value for Exponential Distribution
    argument: ``--inLambda %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inField: ('Uniform' or 'Normal' or 'Exponential')
    Field
    argument: ``--inField %s``
xDefaultMem: (an integer (int or long))
    Set default maximum heap size
    argument: ``-xDefaultMem %d``
inMinimum: (an integer (int or long))
    Minimum Value
    argument: ``--inMinimum %d``
xPrefExt: ('nrrd')
    Output File Type
    argument: ``--xPrefExt %s``
inSize4: (an integer (int or long))
    Size of Volume in t direction
    argument: ``--inSize4 %d``
```

Outputs:

```
outRand1: (an existing file name)
    Rand1
```


66.1 interfaces.mixins.reporting

66.1.1 ReportCapableInterface

[Link to code](#)

Mixin to enable reporting for Nipype interfaces

Inputs:

None

Outputs:

None

67.1 interfaces.mne.base

67.1.1 WatershedBEM

[Link to code](#)

Wraps the executable 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)
    argument: ``--subject %s``
subjects_dir: (an existing directory name, nipy default value: )
    Path to Freesurfer subjects directory

[Optional]
volume: ('T1' or 'aparc+aseg' or 'aseg' or 'brain' or 'orig' or
    'brainmask' or 'ribbon', nipy default value: T1)
    The volume from the "mri" directory to use (defaults to T1)
    argument: ``--volume %s``
overwrite: (a boolean, nipy default value: True)
    Overwrites the existing files
    argument: ``--overwrite``
args: (a unicode string)
```

(continues on next page)

(continued from previous page)

```
Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
Environment variables
atlas_mode: (a boolean)
            Use atlas mode for registration (default: no rigid alignment)
argument: ``--atlas``
```

Outputs:

```
brain_surface: (an existing file name)
               Brain surface (in Freesurfer format)
outer_skin_surface: (an existing file name)
                   Outer skin surface (in Freesurfer format)
fif_file: (an existing file name)
          "fif" format file for EEG processing in MNE
mesh_files: (a list of items which are an existing file name)
            Paths to the output meshes (brain, inner skull, outer skull, outer
            skin)
cor_files: (a list of items which are an existing file name)
           "COR" format files
outer_skull_surface: (an existing file name)
                    Outer skull surface (in Freesurfer format)
inner_skull_surface: (an existing file name)
                    Inner skull surface (in Freesurfer format)
```

68.1 interfaces.mrtrix.convert

68.1.1 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]
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
image_file: (an existing file name)
    The image the tracks were generated from
registration_image_file: (an existing file name)
    The final image the tracks should be registered to.
```

Outputs:

```
out_file: (an existing file name)
```

68.1.2 read_mrtrix_header()

[Link to code](#)

68.1.3 read_mrtrix_streamlines()

[Link to code](#)

68.1.4 read_mrtrix_tracks()

[Link to code](#)

68.1.5 transform_to_affine()

[Link to code](#)

68.2 interfaces.mrtrix.preprocess

68.2.1 DWI2Tensor

[Link to code](#)

Wraps the executable command dwi2tensor.

Converts diffusion-weighted images to tensor images.

Example

```
>>> import nipy.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
        argument: ``%s``, position: -2

[Optional]
debug: (a boolean)
       Display debugging messages.
       argument: ``-debug``, position: 1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
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()
              argument: ``-grad %s``, position: 2
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.
                        argument: ``-ignoreslices %s``, position: 2
quiet: (a boolean)
       Do not display information messages or progress status.
       argument: ``-quiet``, position: 1
out_filename: (a file name)
```

(continues on next page)

(continued from previous page)

```

    Output tensor filename
    argument: ``%s``, position: -1
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.
    argument: ``-ignorevolumes %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

```

Outputs:

```

tensor: (an existing file name)
    path/name of output diffusion tensor image

```

68.2.2 Erode[Link to code](#)

Wraps the executable command erode.

Erode (or dilates) a mask (i.e. binary) image

Example

```

>>> import nipype.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
    argument: ``%s``, position: -2

[Optional]
number_of_passes: (an integer (int or long))
    the number of passes (default: 1)
    argument: ``-npass %s``
dilate: (a boolean)
    Perform dilation rather than erosion
    argument: ``-dilate``, position: 1
debug: (a boolean)
    Display debugging messages.
    argument: ``-debug``, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
quiet: (a boolean)
    Do not display information messages or progress status.
    argument: ``-quiet``, 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: {})

```

(continues on next page)

(continued from previous page)

```

Environment variables
out_filename: (a file name)
Output image filename
argument: ``%s``, position: -1

```

Outputs:

```

out_file: (an existing file name)
the output image

```

68.2.3 GenerateWhiteMatterMask

[Link to code](#)Wraps the executable 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]
in_file: (an existing file name)
Diffusion-weighted images
argument: ``%s``, position: -3
binary_mask: (an existing file name)
Binary brain mask
argument: ``%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
argument: ``-grad %s``, position: 1

[Optional]
out_WMProb_filename: (a file name)
Output WM probability image filename
argument: ``%s``, position: -1
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
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)
argument: ``-margin %s``
environ: (a dictionary with keys which are a bytes or None or a value
of class 'str' and with values which are a bytes or None or a
value of class 'str', nipy default value: {})
Environment variables

```

Outputs:


```
WMprobabilitymap: (an existing file name)
    WMprobabilitymap
```

68.2.4 MRConvert

[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -2

[Optional]
resample: (a float)
    Apply scaling to the intensity values.
    argument: ``-scale %d``, position: 3
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.
    argument: ``-coord %s``, position: 1
out_filename: (a file name)
    Output filename
    argument: ``%s``, position: -1
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"
replace_NaN_with_zero: (a boolean)
    Replace all NaN values with zero.
    argument: ``-zero``, position: 3
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.
    argument: ``-output %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
extract_at_coordinate: (a list of from 1 to 3 items which are a
```

(continues on next page)

(continued from previous page)

```

        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.
        argument: ``%s``, position: 2
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
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"
        argument: ``-output %s``, position: 2
offset_bias: (a float)
        Apply offset to the intensity values.
        argument: ``-scale %d``, position: 3
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.
        argument: ``-vox %s``, position: 3
prs: (a boolean)
        Assume that the DW gradients are specified in the PRS frame (Siemens
        DICOM only).
        argument: ``-prs``, position: 3

```

Outputs:

```

converted: (an existing file name)
        path/name of 4D volume in voxel order

```

68.2.5 MRMultiply[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: -2

[Optional]
debug: (a boolean)
        Display debugging messages.
        argument: ``-debug``, position: 1
args: (a unicode string)
        Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``
quiet: (a boolean)
    Do not display information messages or progress status.
    argument: ``-quiet``, 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
out_filename: (a file name)
    Output image filename
    argument: ``%s``, position: -1

```

Outputs:

```

out_file: (an existing file name)
    the output image of the multiplication

```

68.2.6 MRTransform[Link to code](#)Wraps the executable 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
    argument: ``%s``, position: -2

[Optional]
replace_transform: (a boolean)
    replace the current transform by that specified, rather than
    applying it to the current transform
    argument: ``-replace``, position: 1
out_filename: (a file name)
    Output image
    argument: ``%s``, position: -1
template_image: (an existing file name)
    Reslice the input image to match the specified template image.
    argument: ``-template %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', nipype default value: {})
    Environment variables
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.
    argument: ``-reference %s``, position: 1

```

(continues on next page)

(continued from previous page)

```

invert: (a boolean)
    Invert the specified transform before using it
    argument: ``-inverse``, position: 1
debug: (a boolean)
    Display debugging messages.
    argument: ``-debug``, position: 1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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.
    argument: ``-flipx``, position: 1
quiet: (a boolean)
    Do not display information messages or progress status.
    argument: ``-quiet``, position: 1
transformation_file: (an existing file name)
    The transform to apply, in the form of a 4x4 ascii file.
    argument: ``-transform %s``, position: 1

```

Outputs:

```

out_file: (an existing file name)
    the output image of the transformation

```

68.2.7 MRtrixViewer[Link to code](#)Wraps the executable 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
    argument: ``%s``, position: -2

[Optional]
quiet: (a boolean)
    Do not display information messages or progress status.
    argument: ``-quiet``, position: 1
debug: (a boolean)
    Display debugging messages.
    argument: ``-debug``, position: 1
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

Additional parameters to the command
argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
Environment variables

```

Outputs:

None

68.2.8 MedianFilter3D

[Link to code](#)

Wraps the executable 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
         argument: ``%s``, position: -2

[Optional]
debug: (a boolean)
       Display debugging messages.
       argument: ``-debug``, position: 1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
quiet: (a boolean)
       Do not display information messages or progress status.
       argument: ``-quiet``, 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
out_filename: (a file name)
              Output image filename
              argument: ``%s``, position: -1

```

Outputs:

```

out_file: (an existing file name)
          the output image

```

68.2.9 Tensor2ApparentDiffusion

[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: -2

[Optional]
debug: (a boolean)
       Display debugging messages.
       argument: ``-debug``, position: 1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
quiet: (a boolean)
       Do not display information messages or progress status.
       argument: ``-quiet``, 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
out_filename: (a file name)
              Output Fractional Anisotropy filename
              argument: ``%s``, position: -1
```

Outputs:

```
ADC: (an existing file name)
     the output image of the major eigenvectors of the diffusion tensor
     image.
```

68.2.10 Tensor2FractionalAnisotropy

[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: -2

[Optional]
debug: (a boolean)
        Display debugging messages.
        argument: ``-debug``, position: 1
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
quiet: (a boolean)
        Do not display information messages or progress status.
        argument: ``-quiet``, 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
out_filename: (a file name)
              Output Fractional Anisotropy filename
              argument: ``%s``, position: -1
```

Outputs:

```
FA: (an existing file name)
     the output image of the major eigenvectors of the diffusion tensor
     image.
```

68.2.11 Tensor2Vector[Link to code](#)

Wraps the executable command `tensor2vector`.

Generates a map of the major eigenvectors of the tensors in each voxel.

Example

```
>>> import nipype.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
        argument: ``%s``, position: -2

[Optional]
debug: (a boolean)
        Display debugging messages.
        argument: ``-debug``, position: 1
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
quiet: (a boolean)
```

(continues on next page)

(continued from previous page)

```

    Do not display information messages or progress status.
    argument: ``-quiet``, 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
out_filename: (a file name)
    Output vector filename
    argument: ``%s``, position: -1

```

Outputs:

```

vector: (an existing file name)
    the output image of the major eigenvectors of the diffusion tensor
    image.

```

68.2.12 Threshold[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: -2

[Optional]
quiet: (a boolean)
    Do not display information messages or progress status.
    argument: ``-quiet``, position: 1
invert: (a boolean)
    Invert output binary mask
    argument: ``-invert``, position: 1
debug: (a boolean)
    Display debugging messages.
    argument: ``-debug``, position: 1
replace_zeros_with_NaN: (a boolean)
    Replace all zero values with NaN
    argument: ``-nan``, position: 1
absolute_threshold_value: (a float)
    Specify threshold value as absolute intensity.
    argument: ``-abs %s``
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

        Additional parameters to the command
        argument: ``%s``
percentage_threshold_value: (a float)
        Specify threshold value as a percentage of the peak intensity in the
        input image.
        argument: ``-percent %s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
out_filename: (a file name)
        The output binary image mask.
        argument: ``%s``, position: -1

```

Outputs:

```

out_file: (an existing file name)
        The output binary image mask.

```

68.3 interfaces.mrtrix.tensors

68.3.1 ConstrainedSphericalDeconvolution

[Link to code](#)Wraps the executable 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] $\text{Re} \{Y(2,2)\}$
- [6] $\text{Im} \{Y(4,4)\}$
- [7] $\text{Im} \{Y(4,3)\}$

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]
response_file: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

        the diffusion-weighted signal response function for a single fibre
        population (see EstimateResponse)
        argument: ``%s``, position: -2
in_file: (an existing file name)
        diffusion-weighted image
        argument: ``%s``, position: -3

[Optional]
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.
        argument: ``-lmax %s``
out_filename: (a file name)
        Output filename
        argument: ``%s``, position: -1
directions_file: (an existing file name)
        a text file containing the [ e1 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)
        argument: ``-directions %s``, position: -2
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)
        argument: ``-threshold %s``
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
        argument: ``-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
lambda_value: (a float)
        the regularisation parameter lambda that controls the strength of
        the constraint (default = 1.0).
        argument: ``-lambda %s``
debug: (a boolean)
        Display debugging messages.
        argument: ``-debug``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
normalise: (a boolean)
        normalise the DW signal to the b=0 image
        argument: ``-normalise``, position: 3
mask_image: (an existing file name)
        only perform computation within the specified binary brain mask
        image
        argument: ``-mask %s``, position: 2
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

```

(continues on next page)

(continued from previous page)

```

    0 ]).
    argument: ``-filter %s``, position: -2
iterations: (an integer (int or long))
    the maximum number of iterations to perform for each voxel (default
    = 50)
    argument: ``-niter %s``

```

Outputs:

```

spherical_harmonics_image: (an existing file name)
    Spherical harmonics image

```

68.3.2 DWI2SphericalHarmonicsImage[Link to code](#)

Wraps the executable 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] $\text{Re} \{Y(2,1)\}$
- [5] $\text{Re} \{Y(2,2)\}$
- [6] $\text{Im} \{Y(4,4)\}$
- [7] $\text{Im} \{Y(4,3)\}$

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]
in_file: (an existing file name)
    Diffusion-weighted images
    argument: ``%s``, position: -2
encoding_file: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    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
    argument: ``-grad %s``, position: 1

[Optional]
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.
    argument: ``-lmax %s``
out_filename: (a file name)
    Output filename
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
normalise: (a boolean)
    normalise the DW signal to the b=0 image
    argument: ``-normalise``, 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

```

Outputs:

```

spherical_harmonics_image: (an existing file name)
    Spherical harmonics image

```

68.3.3 Directions2Amplitude[Link to code](#)

Wraps the executable 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.
    argument: ``%s``, position: -2

[Optional]
num_peaks: (an integer (int or long))
    the number of peaks to extract (default is 3)
    argument: ``-num %s``

```

(continues on next page)

(continued from previous page)

```

out_file: (a file name)
    the output amplitudes image
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
peaks_image: (an existing file name)
    the program will try to find the peaks that most closely match those
    in the image provided
    argument: ``-peaks %s``
display_info: (a boolean)
    Display information messages.
    argument: ``-info``
quiet_display: (a boolean)
    do not display information messages or progress status.
    argument: ``-quiet``
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)
    argument: ``-direction %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
display_debug: (a boolean)
    Display debugging messages.
    argument: ``-debug``

```

Outputs:

```

out_file: (an existing file name)
    amplitudes image

```

68.3.4 EstimateResponseForSH[Link to code](#)Wraps the executable 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]
mask_image: (an existing file name)
    only perform computation within the specified binary brain mask

```

(continues on next page)

(continued from previous page)

```

        image
        argument: ``%s``, position: -2
in_file: (an existing file name)
        Diffusion-weighted images
        argument: ``%s``, position: -3
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
        argument: ``-grad %s``, position: 1

[Optional]
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.
        argument: ``-lmax %s``
debug: (a boolean)
        Display debugging messages.
        argument: ``-debug``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
normalise: (a boolean)
        normalise the DW signal to the b=0 image
        argument: ``-normalise``
quiet: (a boolean)
        Do not display information messages or progress status.
        argument: ``-quiet``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
out_filename: (a file name)
        Output filename
        argument: ``%s``, position: -1

```

Outputs:

```

response: (an existing file name)
        Spherical harmonics image

```

68.3.5 FindShPeaks[Link to code](#)

Wraps the executable 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'

```

(continues on next page)

(continued from previous page)

```
>>> shpeaks.inputs.num_peaks = 2
>>> shpeaks.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    the input image of SH coefficients.
    argument: ``%s``, position: -3
directions_file: (an existing file name)
    the set of directions to use as seeds for the peak finding
    argument: ``%s``, position: -2

[Optional]
peak_threshold: (a float)
    only peak amplitudes greater than the threshold will be considered
    argument: ``-threshold %s``
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)
    argument: ``-direction %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
display_info: (a boolean)
    Display information messages.
    argument: ``-info``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
peaks_image: (an existing file name)
    the program will try to find the peaks that most closely match those
    in the image provided
    argument: ``-peaks %s``
num_peaks: (an integer (int or long))
    the number of peaks to extract (default is 3)
    argument: ``-num %s``
quiet_display: (a boolean)
    do not display information messages or progress status.
    argument: ``-quiet``
display_debug: (a boolean)
    Display debugging messages.
    argument: ``-debug``
out_file: (a file name)
    the output image. Each volume corresponds to the x, y & z component
    of each peak direction vector in turn
    argument: ``%s``, position: -1
```

Outputs:

```
out_file: (an existing file name)
    Peak directions image
```

68.3.6 GenerateDirections

[Link to code](#)

Wraps the executable 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.
    argument: ``%s``, position: -2

[Optional]
quiet_display: (a boolean)
    do not display information messages or progress status.
    argument: ``-quiet``
display_info: (a boolean)
    Display information messages.
    argument: ``-info``
out_file: (a file name)
    the text file to write the directions to, as [ az el ] pairs.
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
power: (a float)
    specify exponent to use for repulsion power law.
    argument: ``-power %s``
niter: (an integer (int or long))
    specify the maximum number of iterations to perform.
    argument: ``-niter %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
display_debug: (a boolean)
    Display debugging messages.
    argument: ``-debug``
```

Outputs:

```
out_file: (an existing file name)
    directions file
```

68.3.7 concat_files()

[Link to code](#)

68.4 interfaces.mrtrix.tracking

68.4.1 DiffusionTensorStreamlineTrack

[Link to code](#)

Wraps the executable 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]
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.
    argument: ``%s``, position: -2
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/mm^2). See FSL2MRtrix
    argument: ``-grad %s``, position: -2

[Optional]
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.
    argument: ``-mask %s``, position: 2
    mutually_exclusive: mask_file, mask_spec
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    argument: ``-minlength %s``
seed_file: (an existing file name)
    seed file
    argument: ``-seed %s``
    mutually_exclusive: seed_file, seed_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)
    argument: ``-include %s``, position: 2
    mutually_exclusive: include_file, include_spec
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.
    argument: ``-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
```

(continues on next page)

(continued from previous page)

```

        value of class 'str', nipy default value: {})
    Environment variables
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
    argument: ``-initdirection %s``
step_size: (a float)
    Set the step size of the algorithm in mm (default is 0.2).
    argument: ``-step %s``
stop: (a boolean)
    stop track as soon as it enters any of the include regions.
    argument: ``-stop``
seed_spec: (a list of from 4 to 4 items which are a float)
    seed specification in mm and radius (x y z r)
    argument: ``-seed %s``, position: 2
    mutually_exclusive: seed_file, seed_spec
cutoff_value: (a float)
    Set the FA or FOD amplitude cutoff for terminating tracks (default
    is 0.1).
    argument: ``-cutoff %s``
exclude_spec: (a list of from 4 to 4 items which are a float)
    exclusion specification in mm and radius (x y z r)
    argument: ``-exclude %s``, position: 2
    mutually_exclusive: exclude_file, exclude_spec
out_file: (a file name)
    output data file
    argument: ``%s``, position: -1
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).
    argument: ``-number %d``
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipy default
    value: DT_STREAM)
    input model type
    argument: ``%s``, position: -3
include_file: (an existing file name)
    inclusion file
    argument: ``-include %s``
    mutually_exclusive: include_file, include_spec
unidirectional: (a boolean)
    Track from the seed point in one direction only (default is to track
    in both directions).
    argument: ``-unidirectional``
mask_file: (an existing file name)
    mask file. Only tracks within mask.
    argument: ``-mask %s``
    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).
    argument: ``-maxnum %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

```

(continues on next page)

(continued from previous page)

```

maximum_tract_length: (a float)
    Sets the maximum length of any track in millimeters (default is 200
    mm).
    argument: ``-length %s``
no_mask_interpolation: (a boolean)
    Turns off trilinear interpolation of mask images.
    argument: ``-nomaskinterp``
initial_cutoff_value: (a float)
    Sets the minimum FA or FOD amplitude for initiating tracks (default
    is twice the normal cutoff).
    argument: ``-initcutoff %s``
exclude_file: (an existing file name)
    exclusion file
    argument: ``-exclude %s``
    mutually_exclusive: exclude_file, exclude_spec
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)
    argument: ``-curvature %s``

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

68.4.2 FilterTracks

[Link to code](#)Wraps the executable command `filter_tracks`.

Use regions-of-interest to select a subset of tracks from a given MRtrix track file.

Example

```

>>> import nipyne.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
    argument: ``%s``, position: -2

[Optional]
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    argument: ``-minlength %s``
out_file: (a file name)
    Output filtered track filename
    argument: ``%s``, position: -1
include_spec: (a list of from 4 to 4 items which are a float)
    inclusion specification in mm and radius (x y z r)
    argument: ``-include %s``, position: 2

```

(continues on next page)

(continued from previous page)

```

        mutually_exclusive: include_file, include_spec
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
include_file: (an existing file name)
        inclusion file
        argument: ``-include %s``
        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.
        argument: ``-invert``
debug: (a boolean)
        Display debugging messages.
        argument: ``-debug``, position: 1
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
no_mask_interpolation: (a boolean)
        Turns off trilinear interpolation of mask images.
        argument: ``-nomaskinterp``
exclude_spec: (a list of from 4 to 4 items which are a float)
        exclusion specification in mm and radius (x y z r)
        argument: ``-exclude %s``, position: 2
        mutually_exclusive: exclude_file, exclude_spec
quiet: (a boolean)
        Do not display information messages or progress status.
        argument: ``-quiet``, position: 1
exclude_file: (an existing file name)
        exclusion file
        argument: ``-exclude %s``
        mutually_exclusive: exclude_file, exclude_spec

```

Outputs:

```

out_file: (an existing file name)
        the output filtered tracks

```

68.4.3 ProbabilisticSphericallyDeconvolutedStreamlineTrack[Link to code](#)

Wraps the executable 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.
    argument: ``%s``, position: -2

[Optional]
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.
    argument: ``-mask %s``, position: 2
    mutually_exclusive: mask_file, mask_spec
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    argument: ``-minlength %s``
seed_file: (an existing file name)
    seed file
    argument: ``-seed %s``
    mutually_exclusive: seed_file, seed_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)
    argument: ``-include %s``, position: 2
    mutually_exclusive: include_file, include_spec
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.
    argument: ``-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
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
    argument: ``-initdirection %s``
step_size: (a float)
    Set the step size of the algorithm in mm (default is 0.2).
    argument: ``-step %s``
maximum_number_of_trials: (an integer (int or long))
    Set the maximum number of sampling trials at each point (only used
    for probabilistic tracking).
    argument: ``-trials %s``
stop: (a boolean)
    stop track as soon as it enters any of the include regions.
    argument: ``-stop``
seed_spec: (a list of from 4 to 4 items which are a float)
    seed specification in mm and radius (x y z r)
    argument: ``-seed %s``, position: 2
    mutually_exclusive: seed_file, seed_spec
cutoff_value: (a float)
    Set the FA or FOD amplitude cutoff for terminating tracks (default
    is 0.1).
    argument: ``-cutoff %s``
exclude_spec: (a list of from 4 to 4 items which are a float)

```

(continues on next page)

(continued from previous page)

```

        exclusion specification in mm and radius (x y z r)
        argument: ``-exclude %s``, position: 2
        mutually_exclusive: exclude_file, exclude_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).
        argument: ``-maxnum %d``
out_file: (a file name)
        output data file
        argument: ``%s``, position: -1
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).
        argument: ``-number %d``
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipy default
        value: DT_STREAM)
        input model type
        argument: ``%s``, position: -3
include_file: (an existing file name)
        inclusion file
        argument: ``-include %s``
        mutually_exclusive: include_file, include_spec
unidirectional: (a boolean)
        Track from the seed point in one direction only (default is to track
        in both directions).
        argument: ``-unidirectional``
mask_file: (an existing file name)
        mask file. Only tracks within mask.
        argument: ``-mask %s``
        mutually_exclusive: mask_file, mask_spec
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
maximum_tract_length: (a float)
        Sets the maximum length of any track in millimeters (default is 200
        mm).
        argument: ``-length %s``
no_mask_interpolation: (a boolean)
        Turns off trilinear interpolation of mask images.
        argument: ``-nomaskinterp``
initial_cutoff_value: (a float)
        Sets the minimum FA or FOD amplitude for initiating tracks (default
        is twice the normal cutoff).
        argument: ``-initcutoff %s``
exclude_file: (an existing file name)
        exclusion file
        argument: ``-exclude %s``
        mutually_exclusive: exclude_file, exclude_spec
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)
        argument: ``-curvature %s``

```

Outputs:

```
tracked: (an existing file name)
         output file containing reconstructed tracts
```

68.4.4 SphericallyDeconvolutedStreamlineTrack

[Link to code](#)

Wraps the executable 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.
         argument: ``%s``, position: -2

[Optional]
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.
           argument: ``-mask %s``, position: 2
           mutually_exclusive: mask_file, mask_spec
minimum_tract_length: (a float)
                      Sets the minimum length of any track in millimeters (default is 10
                      mm).
                      argument: ``-minlength %s``
seed_file: (an existing file name)
           seed file
           argument: ``-seed %s``
           mutually_exclusive: seed_file, seed_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)
              argument: ``-include %s``, position: 2
              mutually_exclusive: include_file, include_spec
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.
                   argument: ``-noprocomputed``
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).
```

(continues on next page)

(continued from previous page)

```

        argument: ``-number %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
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
        argument: ``-initdirection %s``
step_size: (a float)
        Set the step size of the algorithm in mm (default is 0.2).
        argument: ``-step %s``
stop: (a boolean)
        stop track as soon as it enters any of the include regions.
        argument: ``-stop``
seed_spec: (a list of from 4 to 4 items which are a float)
        seed specification in mm and radius (x y z r)
        argument: ``-seed %s``, position: 2
        mutually_exclusive: seed_file, seed_spec
cutoff_value: (a float)
        Set the FA or FOD amplitude cutoff for terminating tracks (default
        is 0.1).
        argument: ``-cutoff %s``
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).
        argument: ``-maxnum %d``
out_file: (a file name)
        output data file
        argument: ``%s``, position: -1
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipy default
        value: DT_STREAM)
        input model type
        argument: ``%s``, position: -3
include_file: (an existing file name)
        inclusion file
        argument: ``-include %s``
        mutually_exclusive: include_file, include_spec
unidirectional: (a boolean)
        Track from the seed point in one direction only (default is to track
        in both directions).
        argument: ``-unidirectional``
mask_file: (an existing file name)
        mask file. Only tracks within mask.
        argument: ``-mask %s``
        mutually_exclusive: mask_file, mask_spec
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
maximum_tract_length: (a float)
        Sets the maximum length of any track in millimeters (default is 200
        mm).
        argument: ``-length %s``
no_mask_interpolation: (a boolean)
        Turns off trilinear interpolation of mask images.
        argument: ``-nomaskinterp``

```

(continues on next page)

(continued from previous page)

```

exclude_spec: (a list of from 4 to 4 items which are a float)
    exclusion specification in mm and radius (x y z r)
    argument: ``-exclude %s``, position: 2
    mutually_exclusive: exclude_file, exclude_spec
initial_cutoff_value: (a float)
    Sets the minimum FA or FOD amplitude for initiating tracks (default
    is twice the normal cutoff).
    argument: ``-initcutoff %s``
exclude_file: (an existing file name)
    exclusion file
    argument: ``-exclude %s``
    mutually_exclusive: exclude_file, exclude_spec
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)
    argument: ``-curvature %s``

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

68.4.5 StreamlineTrack[Link to code](#)

Wraps the executable 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.
    argument: ``%s``, position: -2

[Optional]
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

```

(continues on next page)

(continued from previous page)

```

        terminated when they leave the ROI.
        argument: ``-mask %s``, position: 2
        mutually_exclusive: mask_file, mask_spec
minimum_tract_length: (a float)
    Sets the minimum length of any track in millimeters (default is 10
    mm).
    argument: ``-minlength %s``
seed_file: (an existing file name)
    seed file
    argument: ``-seed %s``
    mutually_exclusive: seed_file, seed_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)
    argument: ``-include %s``, position: 2
    mutually_exclusive: include_file, include_spec
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.
    argument: ``-noprecomputed``
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).
    argument: ``-number %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
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
    argument: ``-initdirection %s``
step_size: (a float)
    Set the step size of the algorithm in mm (default is 0.2).
    argument: ``-step %s``
stop: (a boolean)
    stop track as soon as it enters any of the include regions.
    argument: ``-stop``
seed_spec: (a list of from 4 to 4 items which are a float)
    seed specification in mm and radius (x y z r)
    argument: ``-seed %s``, position: 2
    mutually_exclusive: seed_file, seed_spec
cutoff_value: (a float)
    Set the FA or FOD amplitude cutoff for terminating tracks (default
    is 0.1).
    argument: ``-cutoff %s``
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).
    argument: ``-maxnum %d``
out_file: (a file name)
    output data file
    argument: ``%s``, position: -1
inputmodel: ('DT_STREAM' or 'SD_PROB' or 'SD_STREAM', nipy default
    value: DT_STREAM)

```

(continues on next page)

(continued from previous page)

```

    input model type
    argument: ``%s``, position: -3
include_file: (an existing file name)
    inclusion file
    argument: ``-include %s``
    mutually_exclusive: include_file, include_spec
unidirectional: (a boolean)
    Track from the seed point in one direction only (default is to track
    in both directions).
    argument: ``-unidirectional``
mask_file: (an existing file name)
    mask file. Only tracks within mask.
    argument: ``-mask %s``
    mutually_exclusive: mask_file, mask_spec
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
maximum_tract_length: (a float)
    Sets the maximum length of any track in millimeters (default is 200
    mm).
    argument: ``-length %s``
no_mask_interpolation: (a boolean)
    Turns off trilinear interpolation of mask images.
    argument: ``-nomaskinterp``
exclude_spec: (a list of from 4 to 4 items which are a float)
    exclusion specification in mm and radius (x y z r)
    argument: ``-exclude %s``, position: 2
    mutually_exclusive: exclude_file, exclude_spec
initial_cutoff_value: (a float)
    Sets the minimum FA or FOD amplitude for initiating tracks (default
    is twice the normal cutoff).
    argument: ``-initcutoff %s``
exclude_file: (an existing file name)
    exclusion file
    argument: ``-exclude %s``
    mutually_exclusive: exclude_file, exclude_spec
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)
    argument: ``-curvature %s``

```

Outputs:

```

tracked: (an existing file name)
    output file containing reconstructed tracts

```

68.4.6 Tracks2Prob[Link to code](#)

Wraps the executable 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 nipy.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
        argument: ``%s``, position: -2

[Optional]
resample: (a float)
        resample the tracks at regular intervals using Hermite
        interpolation. If omitted, the program will select an appropriate
        interpolation factor automatically.
        argument: ``-resample %d``, position: 3
out_filename: (a file name)
        output data file
        argument: ``%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"
                  argument: ``-datatype %s``, position: 2
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)
        argument: ``-template %s``, position: 1
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
colour: (a boolean)
        add colour to the output image according to the direction of the
        tracks.
        argument: ``-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', nipy 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.
        argument: ``-fraction``, position: 3
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.
        argument: ``-vox %s``, position: 2
```

Outputs:

```
tract_image: (an existing file name)
              Output tract count or track density image
```

69.1 interfaces.mrtrix3.base

69.1.1 MRTrix3Base

[Link to code](#)

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
```

Outputs:

```
None
```

69.2 interfaces.mrtrix3.connectivity

69.2.1 BuildConnectome

[Link to code](#)

Wraps the executable command `tck2connectome`.

Generate a connectome matrix from a streamlines file and a node parcellation image

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> mat = mrt.BuildConnectome()
>>> mat.inputs.in_file = 'tracks.tck'
>>> mat.inputs.in_parcel = 'aparc+aseg.nii'
```

(continues on next page)

(continued from previous page)

```
>>> mat.cmdline
'tck2connectome tracks.tck aparc+aseg.nii connectome.csv'
>>> mat.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    input tractography
    argument: ``%s``, position: -3
out_file: (a file name, nipy default value: connectome.csv)
    output file after processing
    argument: ``%s``, position: -1

[Optional]
vox_lookup: (a boolean)
    use a simple voxel lookup value at each streamline endpoint
    argument: ``-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)
    argument: ``-zero_diagonal``
in_parcc: (an existing file name)
    parcellation file
    argument: ``%s``, position: -2
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)
    argument: ``-keep_unassigned``
in_weights: (an existing file name)
    specify a text scalar file containing the streamline weights
    argument: ``-tck_weights_in %s``
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).
    argument: ``-assignment_reverse_search %f``
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.
    argument: ``-assignment_forward_search %f``
in_scalar: (an existing file name)
    provide the associated image for the mean_scalar metric
    argument: ``-image %s``
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.
    argument: ``-assignment_radial_search %f``
metric: ('count' or 'meanlength' or 'invlength' or 'invnodevolume' or
    'mean_scalar' or 'invlength_invnodevolume')
    specify the edge weight metric
```

(continues on next page)

(continued from previous page)

```

        argument: ``-metric %s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
nthreads: (an integer (int or long))
        number of threads. if zero, the number of available cpus will be
        used
        argument: ``-nthreads %d``

```

Outputs:

```

out_file: (an existing file name)
        the output response file

```

69.2.2 LabelConfig[Link to code](#)

Wraps the executable 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
        argument: ``%s``, position: -3
out_file: (a file name, nipy default value: parcellation.mif)
        output file after processing
        argument: ``%s``, position: -1

[Optional]
lut_itksnap: (a file name)
        get information from an ITK - SNAP lookup table(this includes the
        IIT atlas file "LUT_GM.txt")
        argument: ``-lut_itksnap %s``
in_config: (an existing file name)
        connectome configuration file
        argument: ``%s``, position: -2
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
lut_basic: (a file name)

```

(continues on next page)

(continued from previous page)

```

        get information from a basic lookup table consisting of index / name
        pairs
        argument: ``-lut_basic %s``
lut_fs: (a file name)
        get information from a FreeSurfer lookup table(typically
        "FreeSurferColorLUT.txt")
        argument: ``-lut_freesurfer %s``
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.
        argument: ``-spine %s``
lut_aal: (a file name)
        get information from the AAL lookup table (typically
        "ROI_MNI_V4.txt")
        argument: ``-lut_aal %s``
nthreads: (an integer (int or long))
        number of threads. if zero, the number of available cpus will be
        used
        argument: ``-nthreads %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

```

Outputs:

```

out_file: (an existing file name)
        the output response file

```

69.2.3 LabelConvert[Link to code](#)

Wraps the executable command labelconvert.

Re-configure parcellation to be incrementally defined.

Example

```

>>> import nipy.interfaces.mrtrix3 as mrt
>>> labels = mrt.LabelConvert()
>>> labels.inputs.in_file = 'aparc+aseg.nii'
>>> labels.inputs.in_config = 'mrtrix3_labelconfig.txt'
>>> labels.inputs.in_lut = 'FreeSurferColorLUT.txt'
>>> labels.cmdline
'labelconvert aparc+aseg.nii FreeSurferColorLUT.txt mrtrix3_labelconfig.txt_
↳parcellation.mif'
>>> labels.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input anatomical image
        argument: ``%s``, position: -4
out_file: (a file name, nipy default value: parcellation.mif)
        output file after processing

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``, position: -1
in_lut: (an existing file name)
    get information from a basic lookup table consisting of index / name
    pairs
    argument: ``%s``, position: -3

[Optional]
in_config: (an existing file name)
    connectome configuration file
    argument: ``%s``, position: -2
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
num_threads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    argument: ``-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.
    argument: ``-spine %s``
environ: (a dictionary with keys which are a 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

```

Outputs:

```

out_file: (an existing file name)
    the output response file

```

69.3 interfaces.mrtrix3.preprocess

69.3.1 DWIDenoise

[Link to code](#)

Wraps the executable command dwidenoise.

Denoise DWI data and estimate the noise level based on the optimal threshold for PCA.

DWI data denoising and noise map estimation by exploiting data redundancy in the PCA domain using the prior knowledge that the eigenspectrum of random covariance matrices is described by the universal Marchenko Pastur distribution.

Important note: image denoising must be performed as the first step of the image processing pipeline. The routine will fail if interpolation or smoothing has been applied to the data prior to denoising.

Note that this function does not correct for non-Gaussian noise biases.

For more information, see <<https://mrtrix.readthedocs.io/en/latest/reference/commands/dwidenoise.html>>

Example

```

>>> import nipype.interfaces.mrtrix3 as mrt
>>> denoise = mrt.DWIDenoise()
>>> denoise.inputs.in_file = 'dwi.mif'
>>> denoise.inputs.mask = 'mask.mif'
>>> denoise.cmdline

```

(continues on next page)

(continued from previous page)

```
'dwidenoise -mask mask.mif dwi.mif dwi_denoised.mif'
>>> denoise.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        input DWI image
        argument: ``%s``, position: -2

[Optional]
mask: (an existing file name)
      mask image
      argument: ``-mask %s``, position: 1
in_bval: (an existing file name)
         bvals file in FSL format
in_bvec: (an existing file name)
         bvecs file in FSL format
         argument: ``-fslgrad %s %s``
grad_file: (an existing file name)
           dw gradient scheme (MRtrix format)
           argument: ``-grad %s``
out_file: (a file name)
          the output denoised DWI image
          argument: ``%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
noise: (a file name)
       noise map
       argument: ``-noise %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%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).
            argument: ``-bvalue_scaling %s``
extent: (a tuple of the form: (an integer (int or long), an integer
                              (int or long), an integer (int or long)))
        set the window size of the denoising filter. (default = 5,5,5)
        argument: ``-extent %d,%d,%d``
nthreads: (an integer (int or long))
          number of threads. if zero, the number of available cpus will be
          used
          argument: ``-nthreads %d``
grad_fsl: (a tuple of the form: (an existing file name, an existing
                                file name))
          (bvecs, bvals) dw gradient scheme (FSL format)
          argument: ``-fslgrad %s %s``
```

Outputs:

```
out_file: (an existing file name)
          the output denoised DWI image
```

69.3.2 ResponseSD

[Link to code](#)

Wraps the executable command `dwi2response`.

Estimate response function(s) for spherical deconvolution using the specified algorithm.

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> resp = mrt.ResponseSD()
>>> resp.inputs.in_file = 'dwi.mif'
>>> resp.inputs.algorithm = 'tournier'
>>> resp.inputs.grad_fsl = ('bvecs', 'bvals')
>>> resp.cmdline
'dwi2response tournier -fslgrad bvecs bvals -lmax 8 dwi.mif wm.txt'
>>> resp.run()
```

We can also pass in multiple harmonic degrees in the case of multi-shell >>> resp.inputs.max_sh = [6,8,10]
>>> resp.cmdline 'dwi2response tournier -fslgrad bvecs bvals -lmax 6,8,10 dwi.mif wm.txt'

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input DWI image
         argument: ``%s``, position: -5
algorithm: ('msmt_5tt' or 'dhollander' or 'tournier' or 'tax')
           response estimation algorithm (multi-tissue)
           argument: ``%s``, position: 1

[Optional]
in_bval: (an existing file name)
         bvals file in FSL format
in_bvec: (an existing file name)
         bvecs file in FSL format
         argument: ``-fslgrad %s %s``
grad_file: (an existing file name)
           dw gradient scheme (MRTrix format)
           argument: ``-grad %s``
grad_fsl: (a tuple of the form: (an existing file name, an existing
           file name))
           (bvecs, bvals) dw gradient scheme (FSL format)
           argument: ``-fslgrad %s %s``
wm_file: (a file name, nipy default value: wm.txt)
         output WM response text file
         argument: ``%s``, position: -3
csf_file: (a file name)
         output CSF response text file
         argument: ``%s``, position: -1
max_sh: (a list of items which are an integer (int or long), nipy
        default value: [8])
        maximum harmonic degree of response function - single value for
        single-shell response, list for multi-shell response
        argument: ``-lmax %s``
```

(continues on next page)

(continued from previous page)

```

in_mask: (an existing file name)
    provide initial mask image
    argument: ``-mask %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
mtt_file: (a file name)
    input 5tt image
    argument: ``%s``, position: -4
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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).
    argument: ``-bvalue_scaling %s``
gm_file: (a file name)
    output GM response text file
    argument: ``%s``, position: -2
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    argument: ``-nthreads %d``

```

Outputs:

```

gm_file: (a file name)
    output GM response text file
    argument: ``%s``
csf_file: (a file name)
    output CSF response text file
    argument: ``%s``
wm_file: (a file name)
    output WM response text file
    argument: ``%s``

```

69.4 interfaces.mrtrix3.reconst

69.4.1 EstimateFOD

[Link to code](#)Wraps the executable command `dwi2fod`.

Estimate fibre orientation distributions from diffusion data using spherical deconvolution

Example

```

>>> import nipy.interfaces.mrtrix3 as mrt
>>> fod = mrt.EstimateFOD()
>>> fod.inputs.algorithm = 'csd'
>>> fod.inputs.in_file = 'dwi.mif'
>>> fod.inputs.wm_txt = 'wm.txt'

```

(continues on next page)

(continued from previous page)

```
>>> fod.inputs.grad_fsl = ('bvecs', 'bvals')
>>> fod.cmdline
'dwi2fod -fslgrad bvecs bvals -lmax 8 csd dwi.mif wm.txt wm.mif gm.mif csf.mif'
>>> fod.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input DWI image
         argument: ``%s``, position: -7
wm_txt: (a file name)
         WM response text file
         argument: ``%s``, position: -6
wm_odf: (a file name, nipy default value: wm.mif)
         output WM ODF
         argument: ``%s``, position: -5
algorithm: ('csd' or 'msmt_csd')
           FOD algorithm
           argument: ``%s``, position: -8

[Optional]
gm_txt: (a file name)
         GM response text file
         argument: ``%s``, position: -4
in_bval: (an existing file name)
         bvals file in FSL format
shell: (a list of items which are a float)
        specify one or more dw gradient shells
        argument: ``-shell %s``
max_sh: (an integer (int or long), nipy default value: 8)
        maximum harmonic degree of response function
        argument: ``-lmax %d``
grad_file: (an existing file name)
            dw gradient scheme (MRTrix format)
            argument: ``-grad %s``
grad_fsl: (a tuple of the form: (an existing file name, an existing
            file name))
            (bvecs, bvals) dw gradient scheme (FSL format)
            argument: ``-fslgrad %s %s``
gm_odf: (a file name, nipy default value: gm.mif)
         output GM ODF
         argument: ``%s``, position: -3
csf_odf: (a file name, nipy default value: csf.mif)
         output CSF ODF
         argument: ``%s``, position: -1
in_bvec: (an existing file name)
         bvecs file in FSL format
         argument: ``-fslgrad %s %s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
mask_file: (an existing file name)
            mask image
            argument: ``-mask %s``
args: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

        Additional parameters to the command
        argument: ``%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).
    argument: ``-bvalue_scaling %s``
csf_txt: (a file name)
    CSF response text file
    argument: ``%s``, position: -2
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.
    argument: ``-directions %s``
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    argument: ``-nthreads %d``

```

Outputs:

```

gm_odf: (a file name)
    output GM ODF
    argument: ``%s``
csf_odf: (a file name)
    output CSF ODF
    argument: ``%s``
wm_odf: (a file name)
    output WM ODF
    argument: ``%s``

```

69.4.2 FitTensor[Link to code](#)Wraps the executable 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)

```

(continues on next page)

(continued from previous page)

```

    input diffusion weighted images
    argument: ``%s``, position: -2
out_file: (a file name, nipy default value: dti.mif)
    the output diffusion tensor image
    argument: ``%s``, position: -1

[Optional]
in_bval: (an existing file name)
    bvals file in FSL format
in_bvec: (an existing file name)
    bvecs file in FSL format
    argument: ``-fslgrad %s %s``
grad_file: (an existing file name)
    dw gradient scheme (MRTrix format)
    argument: ``-grad %s``
in_mask: (an existing file name)
    only perform computation within the specified binary brain mask
    image
    argument: ``-mask %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
method: ('nonlinear' or 'loglinear' or 'sech' or 'rician')
    select method used to perform the fitting
    argument: ``-method %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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).
    argument: ``-bvalue_scaling %s``
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
    argument: ``-regularisation %f``
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    argument: ``-nthreads %d``
grad_fsl: (a tuple of the form: (an existing file name, an existing
    file name))
    (bvecs, bvals) dw gradient scheme (FSL format)
    argument: ``-fslgrad %s %s``

```

Outputs:

```

out_file: (an existing file name)
    the output DTI file

```

69.5 interfaces.mrtrix3.tracking

69.5.1 Tractography

[Link to code](#)

Wraps the executable command `tckgen`.

Performs streamlines tractography after selecting the appropriate algorithm.

Example

```
>>> import nipy.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 -samples 4 -output_seeds out_seeds.nii.gz -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
         argument: ``%s``, position: -2
out_file: (a file name, nipy default value: tracked.tck)
         output file containing tracks
         argument: ``%s``, position: -1

[Optional]
in_bval: (an existing file name)
         bvals file in FSL format
grad_fsl: (a tuple of the form: (an existing file name, an existing
         file name))
         (bvecs, bvals) dw gradient scheme (FSL format)
         argument: ``-fslgrad %s %s``
cutoff: (a float)
         set the FA or FOD amplitude cutoff for terminating tracks (default
         is 0.1)
         argument: ``-cutoff %f``
angle: (a float)
         set the maximum angle between successive steps (default is 90deg x
         stepsize / voxelsize)
         argument: ``-angle %f``
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
         argument: ``-algorithm %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
         argument: ``-include %s``
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
```

(continues on next page)

(continued from previous page)

```

    point
    argument: ``-max_seed_attempts %d``
crop_at_gmwmi: (a boolean)
    crop streamline endpoints more precisely as they cross the GM-WM
    interface
    argument: ``-crop_at_gmwmi``
backtrack: (a boolean)
    allow tracks to be truncated
    argument: ``-backtrack``
seed_rejection: (an existing file name)
    seed from an image using rejection sampling (higher values = more
    probable to seed from
    argument: ``-seed_rejection %s``
seed_gmwmi: (an existing file name)
    seed from the grey matter - white matter interface (only valid if
    using ACT framework)
    argument: ``-seed_gmwmi %s``
    requires: act_file
n_samples: (an integer (int or long), nipy default value: 4)
    set the number of FOD samples to take per step for the 2nd order
    (iFOD2) method
    argument: ``-samples %d``
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)
    argument: ``-step %f``
max_length: (a float)
    set the maximum length of any track in mm (default is 100 x
    voxelsize)
    argument: ``-maxlength %f``
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.
    argument: ``-seed_dynamic %s``
unidirectional: (a boolean)
    track from the seed point in one direction only (default is to track
    in both directions)
    argument: ``-unidirectional``
out_seeds: (a file name, nipy default value: out_seeds.nii.gz)
    output the seed location of all successful streamlines to a file
    argument: ``-output_seeds %s``
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    argument: ``-nthreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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
    argument: ``-number %d``
bval_scale: ('yes' or 'no')
```

(continues on next page)

(continued from previous page)

```

    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).
    argument: ``-bvalue_scaling %s``
power: (an integer (int or long))
    raise the FOD to the power specified (default is 1/nsamples)
    argument: ``-power %d``
noprecompt: (a boolean)
    do NOT pre-compute legendre polynomial values. Warning: this will
    slow down the algorithm by a factor of approximately 4
    argument: ``-noprecomputed``
seed_sphere: (a tuple of the form: (a float, a float, a float, a
    float))
    spherical seed
    argument: ``-seed_sphere %f,%f,%f,%f``
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
    argument: ``-exclude %s``
grad_file: (an existing file name)
    dw gradient scheme (MRtrix format)
    argument: ``-grad %s``
sph_trait: (a tuple of the form: (a float, a float, a float, a
    float))
    argument: ``%f,%f,%f,%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
seed_image: (an existing file name)
    seed streamlines entirely at random within mask
    argument: ``-seed_image %s``
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
    argument: ``-act %s``
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)
    argument: ``-seed_grid_per_voxel %s %d``
    mutually_exclusive: seed_image, seed_rnd_voxel
min_length: (a float)
    set the minimum length of any track in mm (default is 5 x voxelsize)
    argument: ``-minlength %f``
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
    argument: ``-seed_random_per_voxel %s %d``
    mutually_exclusive: seed_image, seed_grid_voxel
use_rk4: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        use 4th-order Runge-Kutta integration (slower, but eliminates
        curvature overshoot in 1st-order deterministic methods)
        argument: ``-rk4``
n_trials: (an integer (int or long))
        set the maximum number of sampling trials at each point (only used
        for probabilistic tracking)
        argument: ``-trials %d``
cutoff_init: (a float)
        set the minimum FA or FOD amplitude for initiating tracks (default
        is the same as the normal cutoff)
        argument: ``-initcutoff %f``
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
        argument: ``-mask %s``
in_bvec: (an existing file name)
        bvecs file in FSL format
        argument: ``-fslgrad %s %s``
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)
        argument: ``-maxnum %d``
downsample: (a float)
        downsample the generated streamlines to reduce output file size
        argument: ``-downsample %f``
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
        argument: ``-initdirection %f,%f,%f``
stop: (a boolean)
        stop propagating a streamline once it has traversed all include
        regions
        argument: ``-stop``

```

Outputs:

```

out_seeds: (a file name)
        output the seed location of all successful streamlines to a file
out_file: (an existing file name)
        the output filtered tracks

```

69.6 interfaces.mrtrix3.utils

69.6.1 BrainMask

[Link to code](#)

Wraps the executable command dwi2mask.

Convert a mesh surface to a partial volume estimation image

Example

```

>>> import nipy.interfaces.mrtrix3 as mrt
>>> bmsk = mrt.BrainMask()

```

(continues on next page)

(continued from previous page)

```
>>> 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
         argument: ``%s``, position: -2
out_file: (a file name, nipy default value: brainmask.mif)
         output brain mask
         argument: ``%s``, position: -1

[Optional]
in_bval: (an existing file name)
         bvals file in FSL format
in_bvec: (an existing file name)
         bvecs file in FSL format
         argument: ``-fslgrad %s %s``
grad_file: (an existing file name)
         dw gradient scheme (MRtrix format)
         argument: ``-grad %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%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).
            argument: ``-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
nthreads: (an integer (int or long))
         number of threads. if zero, the number of available cpus will be
         used
         argument: ``-nthreads %d``
grad_fsl: (a tuple of the form: (an existing file name, an existing
         file name))
         (bvecs, bvals) dw gradient scheme (FSL format)
         argument: ``-fslgrad %s %s``
```

Outputs:

```
out_file: (an existing file name)
         the output response file
```

69.6.2 ComputeTDI[Link to code](#)Wraps the executable 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: Dholander, 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
         argument: ``%s``, position: -2

[Optional]
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
          argument: ``-contrast %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)
          argument: ``-map_zero``
out_file: (a file name, nipy default value: tdi.mif)
          output TDI file
          argument: ``%s``, position: -1
reference: (an existing file name)
          a referenceimage to be used as template
```

(continues on next page)

(continued from previous page)

```

        argument: ``-template %s``
stat_vox: ('sum' or 'min' or 'mean' or 'max')
    define the statistic for choosing the finalvoxel intensities for a
    given contrast
    argument: ``-stat_vox %s``
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
    argument: ``-fwhm_tck %f``
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.
    argument: ``-stat_tck %s``
max_tod: (an integer (int or long))
    generate a Track Orientation Distribution (TOD) in each voxel.
    argument: ``-tod %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
ends_only: (a boolean)
    only map the streamline endpoints to the image
    argument: ``-ends_only``
dixel: (a file name)
    map streamlines todixels within each voxel. Directions are stored
    as azimuth elevation pairs.
    argument: ``-dixel %s``
data_type: ('float' or 'unsigned int')
    specify output image data type
    argument: ``-datatype %s``
upsample: (an integer (int or long))
    upsample the tracks by some ratio using Hermite interpolation before
    mapping
    argument: ``-upsample %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
tck_weights: (an existing file name)
    specify a text scalar file containing the streamline weights
    argument: ``-tck_weights_in %s``
use_dec: (a boolean)
    perform mapping in DEC space
    argument: ``-dec``
in_map: (an existing file name)
    provide the scalar image map for generating images with 'scalar_map'
    contrasts, or the SHs image for fod_amp
    argument: ``-image %s``
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)
    argument: ``-precise``
vox_size: (a list of items which are an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

voxel dimensions
argument: ``-vox %s``
nthreads: (an integer (int or long))
number of threads. if zero, the number of available cpus will be
used
argument: ``-nthreads %d``

```

Outputs:

```

out_file: (a file name)
output TDI file

```

69.6.3 DWIExtract[Link to code](#)

Wraps the executable command dwiextract.

Extract diffusion-weighted volumes, b=0 volumes, or certain shells from a DWI dataset

Example

```

>>> import nipy.interfaces.mrtrix3 as mrt
>>> dwiextract = mrt.DWIExtract()
>>> dwiextract.inputs.in_file = 'dwi.mif'
>>> dwiextract.inputs.bzero = True
>>> dwiextract.inputs.out_file = 'b0vols.mif'
>>> dwiextract.inputs.grad_fsl = ('bvecs', 'bvals')
>>> dwiextract.cmdline
'dwiextract -bzero -fslgrad bvecs bvals dwi.mif b0vols.mif'
>>> dwiextract.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
input image
argument: ``%s``, position: -2
out_file: (a file name)
output image
argument: ``%s``, position: -1

[Optional]
in_bval: (an existing file name)
bvals file in FSL format
in_bvec: (an existing file name)
bvecs file in FSL format
argument: ``-fslgrad %s %s``
grad_file: (an existing file name)
dw gradient scheme (MRTrix format)
argument: ``-grad %s``
grad_fsl: (a tuple of the form: (an existing file name, an existing
file name))
(bvecs, bvals) dw gradient scheme (FSL format)
argument: ``-fslgrad %s %s``
nobzero: (a boolean)
extract non b=0 volumes
argument: ``-no_bzero``

```

(continues on next page)

(continued from previous page)

```

shell: (a list of items which are a float)
       specify one or more gradient shells
       argument: ``-shell %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
bzero: (a boolean)
       extract b=0 volumes
       argument: ``-bzero``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%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).
           argument: ``-bvalue_scaling %s``
singleshell: (a boolean)
            extract volumes with a specific shell
            argument: ``-singleshell``
nthreads: (an integer (int or long))
          number of threads. if zero, the number of available cpus will be
          used
          argument: ``-nthreads %d``

```

Outputs:

```

out_file: (an existing file name)
         output image

```

69.6.4 Generate5tt[Link to code](#)

Wraps the executable command 5ttgen.

Generate a 5TT image suitable for ACT using the selected algorithm

Example

```

>>> import nipy.interfaces.mrtrix3 as mrt
>>> gen5tt = mrt.Generate5tt()
>>> gen5tt.inputs.in_file = 'T1.nii.gz'
>>> gen5tt.inputs.algorithm = 'fsl'
>>> gen5tt.inputs.out_file = '5tt.mif'
>>> gen5tt.cmdline
'5ttgen fsl T1.nii.gz 5tt.mif'
>>> gen5tt.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        input image
        argument: ``%s``, position: -2

```

(continues on next page)

(continued from previous page)

```

out_file: (a file name)
    output image
    argument: ``%s``, position: -1
algorithm: ('fsl' or 'gif' or 'freesurfer')
    tissue segmentation algorithm
    argument: ``%s``, position: -3

[Optional]
in_bval: (an existing file name)
    bvals file in FSL format
in_bvec: (an existing file name)
    bvecs file in FSL format
    argument: ``-fslgrad %s %s``
grad_file: (an existing file name)
    dw gradient scheme (MRtrix format)
    argument: ``-grad %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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).
    argument: ``-bvalue_scaling %s``
grad_fsl: (a tuple of the form: (an existing file name, an existing
    file name))
    (bvecs, bvals) dw gradient scheme (FSL format)
    argument: ``-fslgrad %s %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    argument: ``-nthreads %d``

```

Outputs:

```

out_file: (an existing file name)
    output image

```

69.6.5 MRConvert[Link to code](#)Wraps the executable command `mrconvert`.

Perform conversion between different file types and optionally extract a subset of the input image

Example

```

>>> import nipy.interfaces.mrtrix3 as mrt
>>> mrconvert = mrt.MRConvert()
>>> mrconvert.inputs.in_file = 'dwi.nii.gz'

```

(continues on next page)

(continued from previous page)

```
>>> mrconvert.inputs.grad_fsl = ('bvecs', 'bvals')
>>> mrconvert.cmdline
'mrconvert -fslgrad bvecs bvals dwi.nii.gz dwi.mif'
>>> mrconvert.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input image
         argument: ``%s``, position: -2
out_file: (a file name, nipy default value: dwi.mif)
         output image
         argument: ``%s``, position: -1

[Optional]
in_bval: (an existing file name)
         bvals file in FSL format
in_bvec: (an existing file name)
         bvecs file in FSL format
         argument: ``-fslgrad %s %s``
grad_file: (an existing file name)
         dw gradient scheme (MRtrix format)
         argument: ``-grad %s``
grad_fsl: (a tuple of the form: (an existing file name, an existing
         file name))
         (bvecs, bvals) dw gradient scheme (FSL format)
         argument: ``-fslgrad %s %s``
scaling: (a list of items which are a float)
         specify the data scaling parameter
         argument: ``-scaling %s``
coord: (a list of items which are a float)
         extract data at the specified coordinates
         argument: ``-coord %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
args: (a unicode string)
         Additional parameters to the command
         argument: ``%s``
vox: (a list of items which are a float)
         change the voxel dimensions
         argument: ``-vox %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).
         argument: ``-bvalue_scaling %s``
axes: (a list of items which are an integer (int or long))
         specify the axes that will be used
         argument: ``-axes %s``
nthreads: (an integer (int or long))
         number of threads. if zero, the number of available cpus will be
         used
```

(continues on next page)

(continued from previous page)

```
argument: ``-nthreads %d``
```

Outputs:

```
out_file: (an existing file name)
          output image
```

69.6.6 MRMath[Link to code](#)Wraps the executable command `mrmath`.

Compute summary statistic on image intensities along a specified axis of a single image

Example

```
>>> import nipy.interfaces.mrtrix3 as mrt
>>> mrmath = mrt.MRMath()
>>> mrmath.inputs.in_file = 'dwi.mif'
>>> mrmath.inputs.operation = 'mean'
>>> mrmath.inputs.axis = 3
>>> mrmath.inputs.out_file = 'dwi_mean.mif'
>>> mrmath.inputs.grad_fsl = ('bvecs', 'bvals')
>>> mrmath.cmdline
'mrmath -axis 3 -fslgrad bvecs bvals dwi.mif mean dwi_mean.mif'
>>> mrmath.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
         input image
         argument: ``%s``, position: -3
operation: ('mean' or 'median' or 'sum' or 'product' or 'rms' or
           'norm' or 'var' or 'std' or 'min' or 'max' or 'absmax' or
           'magmax')
           operation to computer along a specified axis
           argument: ``%s``, position: -2
out_file: (a file name)
          output image
          argument: ``%s``, position: -1

[Optional]
in_bval: (an existing file name)
         bvals file in FSL format
in_bvec: (an existing file name)
         bvecs file in FSL format
         argument: ``-fslgrad %s %s``
axis: (an integer (int or long))
      specfied axis to perform the operation along
      argument: ``-axis %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
args: (a unicode string)
      Additional parameters to the command
```

(continues on next page)

(continued from previous page)

```

        argument: ``%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).
    argument: ``-bvalue_scaling %s``
grad_file: (an existing file name)
    dw gradient scheme (MRtrix format
    argument: ``-grad %s``
nthreads: (an integer (int or long))
    number of threads. if zero, the number of available cpus will be
    used
    argument: ``-nthreads %d``
grad_fsl: (a tuple of the form: (an existing file name, an existing
    file name))
    (bvecs, bvals) dw gradient scheme (FSL format
    argument: ``-fslgrad %s %s``

```

Outputs:

```

out_file: (an existing file name)
    output image

```

69.6.7 Mesh2PVE[Link to code](#)

Wraps the executable 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 = 'surfl.vtk'
>>> m2p.inputs.reference = 'dwi.mif'
>>> m2p.inputs.in_first = 'T1.nii.gz'
>>> m2p.cmdline
'mesh2pve -first T1.nii.gz surfl.vtk dwi.mif mesh2volume.nii.gz'
>>> m2p.run()

```

Inputs:

```

[Mandatory]
reference: (an existing file name)
    input reference image
    argument: ``%s``, position: -2
in_file: (an existing file name)
    input mesh
    argument: ``%s``, position: -3
out_file: (a file name, nipy default value: mesh2volume.nii.gz)
    output file containing SH coefficients
    argument: ``%s``, position: -1

[Optional]

```

(continues on next page)

(continued from previous page)

```

in_first: (an existing file name)
    indicates that the mesh file is provided by FSL FIRST
    argument: ``-first %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a 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

```

Outputs:

```

out_file: (an existing file name)
    the output response file

```

69.6.8 TCK2VTK[Link to code](#)Wraps the executable command `tck2vtk`.

Convert a track file to a vtk format, cave: coordinates are in XYZ coordinates not reference

Example

```

>>> import nipype.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
    argument: ``%s``, position: -2

[Optional]
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.
    argument: ``-image %s``
out_file: (a file name, nipype default value: tracks.vtk)
    output VTK file
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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).
    argument: ``-image %s``

```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
nthreads: (an integer (int or long))
         number of threads. if zero, the number of available cpus will be
         used
         argument: ``-nthreads %d``

```

Outputs:

```

out_file: (a file name)
         output VTK file

```

69.6.9 TensorMetrics[Link to code](#)Wraps the executable 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 -num 1 -fa fa.mif dti.mif'
>>> comp.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
         input DTI image
         argument: ``%s``, position: -1

[Optional]
out_fa: (a file name)
        output FA file
        argument: ``-fa %s``
out_adc: (a file name)
         output ADC file
         argument: ``-adc %s``
component: (a list of items which are any value, nipy default
           value: [1])
           specify the desired eigenvalue/eigenvector(s). Note that several
           eigenvalues can be specified as a number sequence
           argument: ``-num %s``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
modulate: ('FA' or 'none' or 'eval')
          how to modulate the magnitude of the eigenvectors
          argument: ``-modulate %s``
in_mask: (an existing file name)

```

(continues on next page)

(continued from previous page)

```
    only perform computation within the specified binary brain mask
    image
    argument: ``-mask %s``
out_evec: (a file name)
    output selected eigenvector(s) file
    argument: ``-vector %s``
out_eval: (a file name)
    output selected eigenvalue(s) file
    argument: ``-value %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
Environment variables
```

Outputs:

```
out_eval: (a file name)
    output selected eigenvalue(s) file
out_fa: (a file name)
    output FA file
out_adc: (a file name)
    output ADC file
out_evec: (a file name)
    output selected eigenvector(s) file
```


70.1 interfaces.niftyfit.asl

70.1.1 FitAsl

[Link to code](#)

Wraps the executable 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).
    argument: ``-source %s``, position: 1

[Optional]
error_file: (a file name)
    Filename of the CBF error map.
    argument: ``-error %s``
pasl: (a boolean)
    Fit PASL ASL data [default]
    argument: ``-pasl``
wm_t1: (a float)
```

(continues on next page)

(continued from previous page)

```

    T1 of WM [800ms].
    argument: ``-wmT1 %f``
m0map: (a file name)
    Filename of the estimated input M0 map.
    argument: ``-m0map %s``
m0mape: (a file name)
    Filename of the estimated input M0 map error.
    argument: ``-m0mape %s``
dt_inv2: (a float)
    Difference in inversion time per slice [0ms/slice].
    argument: ``-dTinv2 %f``
segstyle: (a boolean)
    Set CBF as [gm,wm] not [wm,gm].
    argument: ``-segstyle``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
pv_threshold: (a boolean)
    Set PV threshold for switching off LSQR [0.05].
    argument: ``-pvthreshold``
syn_file: (a file name)
    Filename of the synthetic ASL data.
    argument: ``-syn %s``
eff: (a float)
    Labelling efficiency [0.99 (pasl), 0.85 (pcasl)], ensure any
    background suppression pulses are included in -eff
    argument: ``-eff %f``
ldd: (a float)
    Labelling Duration [1800ms].
    argument: ``-LDD %f``
seg: (a file name)
    Filename of the 4D segmentation (in ASL space) for L/T1 estimation
    and PV correction {WM,GM,CSF}.
    argument: ``-seg %s``
pcasl: (a boolean)
    Fit PCASL ASL data
    argument: ``-pcasl``
cbf_file: (a file name)
    Filename of the Cerebral Blood Flow map (in ml/100g/min).
    argument: ``-cbf %s``
plasma_coeff: (a float)
    Single plasma/tissue partition coefficient [0.9ml/g].
    argument: ``-L %f``
t1map: (a file name)
    Filename of the estimated input T1 map (in ms).
    argument: ``-t1map %s``
mask: (a file name)
    Filename of image mask.
    argument: ``-mask %s``, position: 2
ir_volume: (a file name)
    Filename of a [1,2,5]s Inversion Recovery volume (T1/M0 fitting
    carried out internally).
    argument: ``-IRvolume %s``
mulgm: (a boolean)
    Multiply CBF by segmentation [Off].
    argument: ``-sig``

```

(continues on next page)

(continued from previous page)

```

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].
    argument: ``-pv0 %d``
wm_ttt: (a float)
    Time to WM [ATT+0ms].
    argument: ``-wmTTT %f``
out: (a float)
    Outlier rejection for multi CL volumes (enter z-score threshold
    (e.g. 2.5)) [off].
    argument: ``-out %f``
pld: (a float)
    Post Labelling Delay [2000ms].
    argument: ``-PLD %f``
gm_plasma: (a float)
    Plasma/GM water partition [0.95ml/g].
    argument: ``-gmL %f``
t_inv1: (a float)
    Saturation pulse time [800ms].
    argument: ``-Tinv1 %f``
dpld: (a float)
    Difference in labelling delay per slice [0.0 ms/slice].
    argument: ``-dPLD %f``
t_inv2: (a float)
    Inversion time [2000ms].
    argument: ``-Tinv2 %f``
wm_plasma: (a float)
    Plasma/WM water partition [0.82ml/g].
    argument: ``-wmL %f``
sig: (a boolean)
    Use sigmoid to estimate L from T1:  $L(T1|gmL,wmL)$  [Off].
    argument: ``-sig``
mul: (a float)
    Multiply CBF by this value (e.g. if CL are mislabelled use -1.0).
    argument: ``-mul %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
t1_art_cmp: (a float)
    T1 of arterial component [1650ms].
    argument: ``-T1a %f``
pv2: (an integer (int or long))
    In plane PV kernel size [3x3].
    argument: ``-pv2 %d``
gm_t1: (a float)
    T1 of GM [1150ms].
    argument: ``-gmT1 %f``
ir_output: (a file name)
    Output of [1,2,5]s Inversion Recovery fitting.
    argument: ``-IRoutput %s``
gm_ttt: (a float)
    Time to GM [ATT+0ms].
    argument: ``-gmTTT %f``
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].
    argument: ``-pv3 %d %d %d``

```

Outputs:

```
error_file: (a file name)
    Filename of the CBF error map.
syn_file: (a file name)
    Filename of the synthetic ASL data.
cbf_file: (a file name)
    Filename of the Cerebral Blood Flow map (in ml/100g/min).
```

70.2 interfaces.niftyfit.base

70.2.1 NiftyFitCommand

[Link to code](#)

Base support interface for NiftyFit commands.

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

None

70.3 interfaces.niftyfit.dwi

70.3.1 DwiTool

[Link to code](#)

Wraps the executable 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 -
↪ vlmap dwi_vlmap.nii.gz'
```

(continues on next page)

(continued from previous page)

Inputs:

```

[Mandatory]
source_file: (a file name)
    The source image containing the fitted model.
    argument: ``-source %s``, position: 1
bval_file: (a file name)
    The file containing the bvalues of the source DWI.
    argument: ``-bval %s``, position: 2

[Optional]
rgbmap_file: (a file name)
    Filename of colour FA map.
    argument: ``-rgbmap %s``
b0_file: (a file name)
    The B0 image corresponding to the source DWI
    argument: ``-b0 %s``, position: 4
mdmap_file: (a file name)
    Filename of MD map/ADC
    argument: ``-mdmap %s``
mono_flag: (a boolean)
    Input is a single exponential to non-directional data [default with
    no b-vectors]
    argument: ``-mono``, position: 6
    mutually_exclusive: ivim_flag, dti_flag, dti_flag2, ball_flag,
        ballv_flag, nod_flag, nodv_flag
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
ballv_flag: (a boolean)
    Input is a ball and stick model with optimised PDD.
    argument: ``-ballv``, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
        ball_flag, nod_flag, nodv_flag
ball_flag: (a boolean)
    Input is a ball and stick model.
    argument: ``-ball``, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
        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
syn_file: (a file name)
    Filename of synthetic image. Requires: bvec_file/b0_file.
    argument: ``-syn %s``
    requires: bvec_file, b0_file
nodv_flag: (a boolean)
    Input is a NODDI model with optimised PDD
    argument: ``-nodv``, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
        ball_flag, ballv_flag, nod_flag
famap_file: (a file name)
    Filename of FA map
    argument: ``-famap %s``
mcmmap_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

        Filename of multi-compartment model parameter map (-ivim,-ball,-nod)
        argument: ``-mcmmap %s``
dti_flag2: (a boolean)
    Input is a tensor model lower triangular
    argument: ``-dti2``, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag,
        ballv_flag, nod_flag, nodv_flag
ivim_flag: (a boolean)
    Inputs is an IVIM model to non-directional data.
    argument: ``-ivim``, position: 6
    mutually_exclusive: mono_flag, dti_flag, dti_flag2, ball_flag,
        ballv_flag, nod_flag, nodv_flag
vlmap_file: (a file name)
    Filename of PDD map [x,y,z]
    argument: ``-vlmap %s``
dti_flag: (a boolean)
    Input is a tensor model diag/off-diag.
    argument: ``-dti``, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag2, ball_flag,
        ballv_flag, nod_flag, nodv_flag
nod_flag: (a boolean)
    Input is a NODDI model
    argument: ``-nod``, position: 6
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, dti_flag2,
        ball_flag, ballv_flag, nodv_flag
mask_file: (a file name)
    The image mask
    argument: ``-mask %s``, position: 5
bvec_file: (a file name)
    The file containing the bvectors of the source DWI.
    argument: ``-bvec %s``, position: 3
logdti_file: (a file name)
    Filename of output logdti map.
    argument: ``-logdti2 %s``
diso_val: (a float)
    Isotropic diffusivity for -nod [3e-3]
    argument: ``-diso %f``
dpr_val: (a float)
    Parallel diffusivity for -nod [1.7e-3].
    argument: ``-dpr %f``

```

Outputs:

```

rgbmap_file: (a file name)
    Filename of colour FA map
syn_file: (a file name)
    Filename of synthetic image
mdmap_file: (a file name)
    Filename of MD map/ADC
famap_file: (a file name)
    Filename of FA map
logdti_file: (a file name)
    Filename of output logdti map
mcmmap_file: (a file name)
    Filename of multi-compartment model parameter map (-ivim,-ball,-nod)
vlmap_file: (a file name)
    Filename of PDD map [x,y,z]

```

70.3.2 FitDwi

[Link to code](#)

Wraps the executable 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 -
↳ famap dwi_famap.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 -vlmap dwi_vlmap.nii.gz'
```

Inputs:

```
[Mandatory]
bvec_file: (a file name)
    The file containing the b-vectors of the source DWI.
    argument: ``-bvec %s``, position: 3
source_file: (a file name)
    The source image containing the dwi data.
    argument: ``-source %s``, position: 1
bval_file: (a file name)
    The file containing the b-values of the source DWI.
    argument: ``-bval %s``, position: 2

[Optional]
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).
    argument: ``-rotsform %d``
wls_flag: (a boolean)
    Use Variational Bayes fitting with known prior (currently identity
    covariance...).
    argument: ``-wls``
mutually_exclusive: gn_flag
mono_flag: (a boolean)
    Fit single exponential to non-directional data [default with no
    b-vectors]
    argument: ``-mono``, position: 4
mutually_exclusive: ivim_flag, dti_flag, ball_flag, ballv_flag,
    nod_flag, nodv_flag
ballv_flag: (a boolean)
    Fit the ball and stick model with optimised PDD.
    argument: ``-ballv``, position: 4
mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag,
    nod_flag, nodv_flag
ball_flag: (a boolean)
```

(continues on next page)

(continued from previous page)

```

    Fit the ball and stick model.
    argument: ``-ball``, position: 4
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, ballv_flag,
        nod_flag, nodv_flag
syn_file: (a file name)
    Filename of synthetic image.
    argument: ``-syn %s``
nodv_flag: (a boolean)
    Fit the NODDI model with optimised PDD
    argument: ``-nodv``, position: 4
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag,
        ballv_flag, nod_flag
famap_file: (a file name)
    Filename of FA map
    argument: ``-famap %s``
tenmap2_file: (a file name)
    Filename of tensor map [lower tri]
    argument: ``-tenmap2 %s``
    requires: dti_flag
wm_t2_val: (a float)
    White matter T2 value [80ms].
    argument: ``-wmT2 %f``
tenmap_file: (a file name)
    Filename of tensor map [diag,offdiag].
    argument: ``-tenmap %s``
    requires: dti_flag
ivim_flag: (a boolean)
    Fit IVIM model to non-directional data.
    argument: ``-ivim``, position: 4
    mutually_exclusive: mono_flag, dti_flag, ball_flag, ballv_flag,
        nod_flag, nodv_flag
mcout: (a file name)
    Filename of mc samples (ascii text file)
    argument: ``-mcout %s``
slice_no: (an integer (int or long))
    Fit to single slice number.
    argument: ``-slice %d``
prior_file: (a file name)
    Filename of parameter priors for -ball and -nod.
    argument: ``-prior %s``
vlmap_file: (a file name)
    Filename of PDD map [x,y,z]
    argument: ``-vlmap %s``
te_value: (a file name)
    Value of TEs (ms).
    argument: ``-TE %s``
    mutually_exclusive: te_file
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
lm_vals: (a tuple of the form: (a float, a float))
    LM parameters (initial value, decrease rate) [100,1.2].
    argument: ``-lm %f %f``
    requires: gn_flag
dpr_val: (a float)
    Parallel diffusivity for -nod [1.7e-3].
    argument: ``-dpr %f``

```

(continues on next page)

(continued from previous page)

```

maxit_val: (an integer (int or long))
    Maximum number of non-linear LSQR iterations [100x2 passes]
    argument: ``-maxit %d``
    requires: gn_flag
cov_file: (a file name)
    Filename of the nc*nc covariance matrix [I]
    argument: ``-cov %s``
error_file: (a file name)
    Filename of parameter error maps.
    argument: ``-error %s``
mdmap_file: (a file name)
    Filename of MD map/ADC
    argument: ``-mdmap %s``
nod_flag: (a boolean)
    Fit the NODDI model
    argument: ``-nod``, position: 4
    mutually_exclusive: mono_flag, ivim_flag, dti_flag, ball_flag,
        ballv_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
rgbmap_file: (a file name)
    Filename of colour-coded FA map
    argument: ``-rgbmap %s``
    requires: dti_flag
perf_thr: (a float)
    Threshold for perfusion/diffusion effects [100].
    argument: ``-perfthreshold %f``
gn_flag: (a boolean)
    Use Gauss-Newton algorithm [Levenberg-Marquardt].
    argument: ``-gn``
    mutually_exclusive: wls_flag
te_file: (a file name)
    Filename of TEs (ms).
    argument: ``-TE %s``
    mutually_exclusive: te_file
diso_val: (a float)
    Isotropic diffusivity for -nod [3e-3]
    argument: ``-diso %f``
swls_val: (a float)
    Use location-weighted least squares for DTI fitting [3x3 Gaussian]
    argument: ``-swls %f``
mcsamples: (an integer (int or long))
    Number of samples to keep [100].
    argument: ``-mcsamples %d``
vb_flag: (a boolean)
    Use Variational Bayes fitting with known prior (currently identity
    covariance...).
    argument: ``-vb``
res_file: (a file name)
    Filename of model residual map.
    argument: ``-res %s``
ten_type: ('lower-tri' or 'diag-off-diag', nipy default value:
    lower-tri)
    Use lower triangular (tenmap2) or diagonal, off-diagonal tensor
    format

```

(continues on next page)

(continued from previous page)

```

mcmap_file: (a file name)
    Filename of multi-compartment model parameter map (-ivim,-ball,-nod)
    argument: ``-mcmap %s``
    requires: nodv_flag
dti_flag: (a boolean)
    Fit the tensor model [default with b-vectors].
    argument: ``-dti``, position: 4
    mutually_exclusive: mono_flag, ivim_flag, ball_flag, ballv_flag,
        nod_flag, nodv_flag
acceptance: (a float)
    Fraction of iterations to accept [0.23].
    argument: ``-accpetance %f``
nodiff_file: (a file name)
    Filename of average no diffusion image.
    argument: ``-nodiff %s``
mask_file: (a file name)
    The image mask
    argument: ``-mask %s``
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.
    argument: ``-voxel %d %d %d``
csf_t2_val: (a float)
    CSF T2 value [400ms].
    argument: ``-csfT2 %f``
mcmaxit: (an integer (int or long))
    Number of iterations to run [10,000].
    argument: ``-mcmaxit %d``

```

Outputs:

```

error_file: (a file name)
    Filename of parameter error maps
res_file: (a file name)
    Filename of model residual map
mdmap_file: (a file name)
    Filename of MD map/ADC
v1map_file: (a file name)
    Filename of PDD map [x,y,z]
mcmap_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)
rgbmap_file: (a file name)
    Filename of colour FA map
syn_file: (a file name)
    Filename of synthetic image
nodiff_file: (a file name)
    Filename of average no diffusion image.
fomap_file: (a file name)
    Filename of FA map
tenmap2_file: (a file name)
    Filename of tensor map [lower tri]
tenmap_file: (a file name)
    Filename of tensor map

```

70.4 interfaces.niftyfit.qt1

70.4.1 FitQt1

[Link to code](#)

Wraps the executable 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 -mcmap TI4D_mcmmap.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.
    argument: ``-source %s``, position: 1

[Optional]
error_file: (a file name)
    Filename of the error map (symmetric matrix, [Diag,OffDiag]).
    argument: ``-error %s``
tis_list: (a file name)
    Filename of list of pre-defined TIs.
    argument: ``-TIlist %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
tis: (a list of items which are a float)
    Inversion times for T1 data [1s,2s,5s].
    argument: ``-TIs %s``, position: 14
comp_file: (a file name)
    Filename of the estimated multi-component T1 map.
    argument: ``-comp %s``
nb_comp: (an integer (int or long))
    Number of components to fit [1] (currently IR/SR only)
    argument: ``-nc %d``, position: 6
m0map_file: (a file name)
    Filename of the estimated input M0 map.
    argument: ``-m0map %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
tr_value: (a float)
    TR Repetition Time [10s!].
    argument: ``-TR %f``, position: 5
```

(continues on next page)

(continued from previous page)

```

gn_flag: (a boolean)
    Use Gauss-Newton algorithm [Levenberg-Marquardt].
    argument: ``-gn``, position: 8
maxit: (an integer (int or long))
    NLSQR iterations [100].
    argument: ``-maxit %d``, position: 11
mcsamples: (an integer (int or long))
    Number of samples to keep [100].
    argument: ``-mcsamples %d``
tl_list: (a file name)
    Filename of list of pre-defined T1s
    argument: ``-Tllist %s``
prior: (an existing file name)
    Filename of parameter prior.
    argument: ``-prior %s``, position: 3
tlmin: (a float)
    Minimum tissue T1 value [400ms].
    argument: ``-Tlmin %f``
blmap: (a file name)
    Filename of B1 estimate for fitting (or include in prior).
    argument: ``-blmap %s``
mask: (an existing file name)
    Filename of image mask.
    argument: ``-mask %s``, position: 2
res_file: (a file name)
    Filename of the model fit residuals
    argument: ``-res %s``
tlmax: (a float)
    Maximum tissue T1 value [4000ms].
    argument: ``-Tlmax %f``
mcmmap_file: (a file name)
    Filename of the estimated output multi-parameter map.
    argument: ``-mcmmap %s``
slice_no: (an integer (int or long))
    Fit to single slice number.
    argument: ``-slice %d``, position: 9
tlmap_file: (a file name)
    Filename of the estimated output T1 map (in ms).
    argument: ``-tlmap %s``
mcout: (a file name)
    Filename of mc samples (ascii text file)
    argument: ``-mcout %s``
syn_file: (a file name)
    Filename of the synthetic ASL data.
    argument: ``-syn %s``
acceptance: (a float)
    Fraction of iterations to accept [0.23].
    argument: ``-acceptance %f``
spgr: (a boolean)
    Spoiled Gradient Echo fitting
    argument: ``-SPGR``
flips: (a list of items which are a float)
    Flip angles
    argument: ``-flips %s``
te_value: (a float)
    TE Echo Time [0ms!].
    argument: ``-TE %f``, position: 4

```

(continues on next page)

(continued from previous page)

```

ir_flag: (a boolean)
    Inversion Recovery fitting [default].
    argument: ``-IR``, position: 13
lm_val: (a tuple of the form: (a float, a float))
    Set LM parameters (initial value, decrease rate) [100,1.2].
    argument: ``-lm %f %f``, position: 7
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.
    argument: ``-voxel %d %d %d``, position: 10
mcmaxit: (an integer (int or long))
    Number of iterations to run [10,000].
    argument: ``-mcmaxit %d``
flips_list: (a file name)
    Filename of list of pre-defined flip angles (deg).
    argument: ``-fliplist %s``
sr_flag: (a boolean)
    Saturation Recovery fitting [default].
    argument: ``-SR``, position: 12

```

Outputs:

```

error_file: (a file name)
    Filename of the error map (symmetric matrix, [Diag,OffDiag])
res_file: (a file name)
    Filename of the model fit residuals
syn_file: (a file name)
    Filename of the synthetic ASL data
mcmap_file: (a file name)
    Filename of the estimated output multi-parameter map
t1map_file: (a file name)
    Filename of the estimated output T1 map (in ms)
m0map_file: (a file name)
    Filename of the m0 map
comp_file: (a file name)
    Filename of the estimated multi-component T1 map.

```


71.1 interfaces.niftyreg.base

71.1.1 NiftyRegCommand

[Link to code](#)

Base support interface for NiftyReg commands.

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
omp_core_val: (an integer (int or long), nipy default value: 1)
              Number of openmp thread to use
              argument: ``-omp %i``
```

Outputs:

```
None
```

71.1.2 get_custom_path()

[Link to code](#)

71.2 interfaces.niftyreg.reg

71.2.1 RegAladin

[Link to code](#)

Wraps the executable 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]
ref_file: (an existing file name)
    The input reference/target image
    argument: ``-ref %s``
flo_file: (an existing file name)
    The input floating/source image
    argument: ``-flo %s``

[Optional]
v_val: (a long integer >= 0)
    Percent of blocks that are active
    argument: ``-pv %d``
rmask_file: (an existing file name)
    The input reference mask
    argument: ``-rmask %s``
smoo_r_val: (a float)
    Amount of smoothing to apply to reference image
    argument: ``-smooR %f``
rig_only_flag: (a boolean)
    Do only a rigid registration
    argument: ``-rigOnly``
ln_val: (a long integer >= 0)
    Number of resolution levels to create
    argument: ``-ln %d``
i_val: (a long integer >= 0)
    Percent of inlier blocks
    argument: ``-pi %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
omp_core_val: (an integer (int or long), nipy default value: 1)
    Number of openmp thread to use
    argument: ``-omp %i``
ref_low_val: (a float)
    Lower threshold value on reference image
    argument: ``-refLowThr %f``
ref_up_val: (a float)
    Upper threshold value on reference image
```

(continues on next page)

(continued from previous page)

```

        argument: ``-refUpThr %f``
cog_flag: (a boolean)
    Use the masks centre of mass to initialise the transformation
    argument: ``-cog``
aff_direct_flag: (a boolean)
    Directly optimise the affine parameters
    argument: ``-affDirect``
lp_val: (a long integer >= 0)
    Number of resolution levels to perform
    argument: ``-lp %d``
res_file: (a file name)
    The affine transformed floating image
    argument: ``-res %s``
verbosity_off_flag: (a boolean)
    Turn off verbose output
    argument: ``-voff``
platform_val: (an integer (int or long))
    Platform index
    argument: ``-platf %i``
maxit_val: (a long integer >= 0)
    Maximum number of iterations
    argument: ``-maxit %d``
fmask_file: (an existing file name)
    The input floating mask
    argument: ``-fmask %s``
in_aff_file: (an existing file name)
    The input affine transformation
    argument: ``-inaff %s``
aff_file: (a file name)
    The output affine matrix file
    argument: ``-aff %s``
flo_up_val: (a float)
    Upper threshold value on floating image
    argument: ``-floUpThr %f``
smoo_f_val: (a float)
    Amount of smoothing to apply to floating image
    argument: ``-smooF %f``
flo_low_val: (a float)
    Lower threshold value on floating image
    argument: ``-floLowThr %f``
nosym_flag: (a boolean)
    Turn off symmetric registration
    argument: ``-noSym``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
nac_flag: (a boolean)
    Use nifti header to initialise transformation
    argument: ``-nac``
gpuid_val: (an integer (int or long))
    Device to use id
    argument: ``-gpuid %i``

```

Outputs:

```

res_file: (a file name)
    The output transformed image

```

(continues on next page)

(continued from previous page)

```

aff_file: (a file name)
    The output affine file
avg_output: (a string)
    Output string in the format for reg_average

```

71.2.2 RegF3D

[Link to code](#)

Wraps the executable 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]
ref_file: (an existing file name)
    The input reference/target image
    argument: ``-ref %s``
flo_file: (an existing file name)
    The input floating/source image
    argument: ``-flo %s``

[Optional]
maxit_val: (a long integer >= 0)
    Maximum number of iterations per level
    argument: ``-maxit %d``
rmask_file: (an existing file name)
    Reference image mask
    argument: ``-rmask %s``
pad_val: (a float)
    Padding value
    argument: ``-pad %f``
be_val: (a float)
    Bending energy value
    argument: ``-be %f``
ln_val: (a long integer >= 0)
    Number of resolution levels to create
    argument: ``-ln %d``
lncc_val: (a float)
    SD of the Gaussian for computing LNCC
    argument: ``--lncc %f``

```

(continues on next page)

(continued from previous page)

```

ref_smooth_val: (a float)
    Smoothing kernel width for reference image
    argument: ``-smooR %f``
incpp_file: (an existing file name)
    The input cpp transformation file
    argument: ``-incpp %s``
omp_core_val: (an integer (int or long), nipy default value: 1)
    Number of openmp thread to use
    argument: ``-omp %i``
rlwth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
    Lower threshold for reference image at the specified time point
    argument: ``-rLwTh %d %f``
flo_smooth_val: (a float)
    Smoothing kernel width for floating image
    argument: ``-smooF %f``
lp_val: (a long integer >= 0)
    Number of resolution levels to perform
    argument: ``-lp %d``
rupth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
    Upper threshold for reference image at the specified time point
    argument: ``-rUpTh %d %f``
fupth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
    Upper threshold for floating image at the specified time point
    argument: ``-fUpTh %d %f``
cpp_file: (a file name)
    The output CPP file
    argument: ``-cpp %s``
no_app_jl_flag: (a boolean)
    Do not approximate the log of jacobian penalty at control points
    only
    argument: ``-noAppJL``
fupth_thr_val: (a float)
    Upper threshold for floating image
    argument: ``--fUpTh %f``
fmask_file: (an existing file name)
    Floating image mask
    argument: ``-fmask %s``
noy_flag: (a boolean)
    Don't optimise in y direction
    argument: ``-noy``
sx_val: (a float)
    Final grid spacing along the x axes
    argument: ``-sx %f``
noconj_flag: (a boolean)
    Use simple GD optimization
    argument: ``-noConj``
nmi_flag: (a boolean)
    use NMI even when other options are specified
    argument: ``--nmi``
ssd_flag: (a boolean)
    Use SSD as the similarity measure
    argument: ``--ssd``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
kld_flag: (a boolean)
    Use KL divergence as the similarity measure

```

(continues on next page)

(continued from previous page)

```

        argument: ``--kld``
aff_file: (an existing file name)
        The input affine transformation file
        argument: ``-aff %s``
fbn_val: (a long integer >= 0)
        Number of bins in the histogram for reference image
        argument: ``--fbn %d``
sy_val: (a float)
        Final grid spacing along the y axes
        argument: ``-sy %f``
smooth_grad_val: (a float)
        Kernel width for smoothing the metric gradient
        argument: ``-smoothGrad %f``
flwth_thr_val: (a float)
        Lower threshold for floating image
        argument: ``--fLwTh %f``
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
        argument: ``-lncc %d %f``
pert_val: (a long integer >= 0)
        Add perturbation steps after each optimization step
        argument: ``-pert %d``
nox_flag: (a boolean)
        Don't optimise in x direction
        argument: ``-nox``
rupth_thr_val: (a float)
        Upper threshold for reference image
        argument: ``--rUpTh %f``
kld2_flag: (a long integer >= 0)
        Use KL divergence as the similarity measure for a given time point
        argument: ``-kld %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
noz_flag: (a boolean)
        Don't optimise in z direction
        argument: ``-noz``
rbn_val: (a long integer >= 0)
        Number of bins in the histogram for reference image
        argument: ``--rbn %d``
flwth2_thr_val: (a tuple of the form: (a long integer >= 0, a float))
        Lower threshold for floating image at the specified time point
        argument: ``-fLwTh %d %f``
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
        argument: ``-fbn %d %d``
res_file: (a file name)
        The output resampled image
        argument: ``-res %s``
rlwth_thr_val: (a float)
        Lower threshold for reference image
        argument: ``--rLwTh %f``
nopy_flag: (a boolean)
        Do not use the multiresolution approach

```

(continues on next page)

(continued from previous page)

```

        argument: ``-nopy``
verbosity_off_flag: (a boolean)
    Turn off verbose output
    argument: ``-voff``
le_val: (a float)
    Linear elasticity penalty term
    argument: ``-le %f``
ssd2_flag: (a long integer >= 0)
    Use SSD as the similarity measure for a given time point
    argument: ``-ssd %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
    argument: ``-rbn %d %d``
jl_val: (a float)
    Log of jacobian of deformation penalty value
    argument: ``-jl %f``
vel_flag: (a boolean)
    Use velocity field integration
    argument: ``-vel``
amc_flag: (a boolean)
    Use additive NMI
    argument: ``-amc``
sz_val: (a float)
    Final grid spacing along the z axes
    argument: ``-sz %f``

```

Outputs:

```

res_file: (a file name)
    The output resampled image
invres_file: (a file name)
    The output inverse res file
invcpp_file: (a file name)
    The output inverse CPP file
avg_output: (a string)
    Output string in the format for reg_average
cpp_file: (a file name)
    The output CPP file

```

71.3 interfaces.niftyreg.regutils

71.3.1 RegAverage

[Link to code](#)

Wraps the executable 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:

```
[Optional]
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
    argument: ``-avg_tran %s``, position: 1
    mutually_exclusive: avg_files, avg_lts_files, demean1_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
    argument: ``-demean2 %s``, position: 1
    mutually_exclusive: avg_files, avg_lts_files, avg_ref_file,
        demean1_ref_file, demean3_ref_file
    requires: warp_files
avg_lts_files: (a list of items which are a file name)
    Robust average of affine transformations
    argument: ``-avg_lts %s``, position: 1
    mutually_exclusive: avg_files, avg_ref_file, demean1_ref_file,
        demean2_ref_file, demean3_ref_file, warp_files
demean1_ref_file: (a file name)
    Average images and demean average image that have affine
    transformations to a common space
    argument: ``-demean1 %s``, position: 1
    mutually_exclusive: avg_files, avg_lts_files, avg_ref_file,
        demean2_ref_file, demean3_ref_file
    requires: warp_files
out_file: (a file name)
    Output file name
    argument: ``%s``, position: 0
avg_files: (a list of items which are a file name)
    Averaging of images/affine transformations
    argument: ``-avg %s``, position: 1
    mutually_exclusive: avg_lts_files, avg_ref_file, demean1_ref_file,
        demean2_ref_file, demean3_ref_file, 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
    argument: ``-demean3 %s``, position: 1
    mutually_exclusive: avg_files, avg_lts_files, avg_ref_file,
        demean1_ref_file, demean2_ref_file
    requires: warp_files
args: (a unicode string)
    Additional parameters to the command
```

(continues on next page)

(continued from previous page)

```

    argument: ``%s``
warp_files: (a list of items which are a file name)
    transformation files and floating image pairs/triplets to the
    reference space
    argument: ``%s``, position: -1
    mutually_exclusive: avg_files, avg_lts_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', nipype default value: {})
    Environment variables
omp_core_val: (an integer (int or long), nipype default value: 1)
    Number of openmp thread to use
    argument: ``-omp %i``

```

Outputs:

```

out_file: (a file name)
    Output file name

```

71.3.2 RegJacobian

[Link to code](#)Wraps the executable 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 nipype.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
    argument: ``-trans %s``

[Optional]
ref_file: (an existing file name)
    Reference/target file (required if specifying CPP transformations.
    argument: ``-ref %s``
out_file: (a file name)
    The output jacobian determinant file name
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
type: ('jac' or 'jacL' or 'jacM', nipype default value: jac)
    Type of jacobian outcome

```

(continues on next page)

(continued from previous page)

```

        argument: ``-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
omp_core_val: (an integer (int or long), nipy default value: 1)
        Number of openmp thread to use
        argument: ``-omp %i``

```

Outputs:

```

out_file: (a file name)
        The output file

```

71.3.3 RegMeasure[Link to code](#)Wraps the executable 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]
ref_file: (an existing file name)
        The input reference/target image
        argument: ``-ref %s``
measure_type: ('ncc' or 'lncc' or 'nmi' or 'ssd')
        Measure of similarity to compute
        argument: ``-s``
flo_file: (an existing file name)
        The input floating/source image
        argument: ``-flo %s``

[Optional]
out_file: (a file name)
        The output text file containing the measure
        argument: ``-out %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})

```

(continues on next page)

(continued from previous page)

```

Environment variables
omp_core_val: (an integer (int or long), nipy default value: 1)
               Number of openmp thread to use
               argument: ``-omp %i``

```

Outputs:

```

out_file: (a file name)
           The output text file containing the measure

```

71.3.4 RegResample[Link to code](#)Wraps the executable 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 nipy.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]
ref_file: (an existing file name)
           The input reference/target image
           argument: ``-ref %s``
flo_file: (an existing file name)
           The input floating/source image
           argument: ``-flo %s``

[Optional]
verbosity_off_flag: (a boolean)
                    Turn off verbose output
                    argument: ``-voff``
out_file: (a file name)
           The output filename of the transformed image
           argument: ``%s``, position: -1
psf_alg: (0 or 1)
          Minimise the matrix metric (0) or the determinant (1) when
          estimating the PSF [0]
          argument: ``-psf_alg %d``
type: ('res' or 'blank', nipy default value: res)
      Type of output
      argument: ``-%s``, position: -2

```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
omp_core_val: (an integer (int or long), nipy default value: 1)
         Number of openmp thread to use
         argument: ``-omp %i``
trans_file: (an existing file name)
         The input transformation file
         argument: ``-trans %s``
inter_val: ('NN' or 'LIN' or 'CUB' or 'SINC')
         Interpolation type
         argument: ``-inter %d``
psf_flag: (a boolean)
         Perform the resampling in two steps to resample an image to a lower
         resolution
         argument: ``-psf``
args: (a unicode string)
         Additional parameters to the command
         argument: ``%s``
tensor_flag: (a boolean)
         Resample Tensor Map
         argument: ``-tensor``
pad_val: (a float)
         Padding value
         argument: ``-pad %f``

```

Outputs:

```

out_file: (a file name)
         The output filename of the transformed image

```

71.3.5 RegTools

[Link to code](#)Wraps the executable 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

```

(continues on next page)

(continued from previous page)

```

        argument: ``-in %s``

[Optional]
iso_flag: (a boolean)
    Make output image isotropic
    argument: ``-iso``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
div_val: (a float or an existing file name)
    Divide the input by image or value
    argument: ``-div %s``
rms_val: (an existing file name)
    Compute the mean RMS between the images
    argument: ``-rms %s``
out_file: (a file name)
    The output file name
    argument: ``-out %s``
smo_g_val: (a tuple of the form: (a float, a float, a float))
    Smooth the input image using a Gaussian kernel
    argument: ``-smoG %f %f %f``
mul_val: (a float or an existing file name)
    Multiply the input by image or value
    argument: ``-mul %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
omp_core_val: (an integer (int or long), nipy default value: 1)
    Number of openmp thread to use
    argument: ``-omp %i``
add_val: (a float or an existing file name)
    Add to the input image or value
    argument: ``-add %s``
thr_val: (a float)
    Binarise the input image with the given threshold
    argument: ``-thr %f``
mask_file: (an existing file name)
    Values outside the mask are set to NaN
    argument: ``-nan %s``
smo_s_val: (a tuple of the form: (a float, a float, a float))
    Smooth the input image using a cubic spline kernel
    argument: ``-smoS %f %f %f``
chg_res_val: (a tuple of the form: (a float, a float, a float))
    Change the resolution of the input image
    argument: ``-chgres %f %f %f``
bin_flag: (a boolean)
    Binarise the input image
    argument: ``-bin``
sub_val: (a float or an existing file name)
    Add to the input image or value
    argument: ``-sub %s``
down_flag: (a boolean)
    Downsample the image by a factor of 2
    argument: ``-down``
noscl_flag: (a boolean)
    Set scale, slope to 0 and 1

```

(continues on next page)

(continued from previous page)

```

        argument: ``-noscl``
inter_val: ('NN' or 'LIN' or 'CUB' or 'SINC')
    Interpolation order to use to warp the floating image
        argument: ``-interp %d``

```

Outputs:

```

out_file: (an existing file name)
    The output file

```

71.3.6 RegTransform

[Link to code](#)Wraps the executable 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:

```

[Optional]
comp_input: (an existing file name)
    compose two transformations
    argument: ``-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
half_input: (an existing file name)
    Half way to the input transformation
    argument: ``-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
refl_file: (an existing file name)
    The input reference/target image
    argument: ``-ref %s``, position: 0
out_file: (a file name)
    transformation file to write
    argument: ``%s``, position: -1
inv_aff_input: (an existing file name)
    Invert an affine transformation
    argument: ``-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

```

(continues on next page)

(continued from previous page)

```

upd_s_form_input: (an existing file name)
    Update s-form using the affine transformation
    argument: ``-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
aff_2_rig_input: (an existing file name)
    Extract the rigid component from affine transformation
    argument: ``-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
def_input: (an existing file name)
    Compute deformation field from transformation
    argument: ``-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
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
    argument: ``-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
inv_nrr_input: (a tuple of the form: (an existing file name, an
    existing file name))
    Invert a non-linear transformation
    argument: ``-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
environ: (a dictionary with keys which are a 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
upd_s_form_input2: (an existing file name)
    Update s-form using the affine transformation
    argument: ``%s``, position: -2
    requires: upd_s_form_input
omp_core_val: (an integer (int or long), nipype default value: 1)
    Number of openmp thread to use
    argument: ``-omp %i``
ref2_file: (an existing file name)
    The input second reference/target image
    argument: ``-ref2 %s``, position: 1
    requires: ref1_file
comp_input2: (an existing file name)
    compose two transformations
    argument: ``%s``, position: -2
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

```

(continues on next page)

(continued from previous page)

```
argument: ``-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
argument: ``-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
args: (a unicode string)
    Additional parameters to the command
argument: ``%s``
disp_input: (an existing file name)
    Compute displacement field from transformation
argument: ``-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
```

Outputs:

```
out_file: (a file name)
    Output File (transformation in any format)
```

72.1 interfaces.niftyseg.base

72.1.1 NiftySegCommand

[Link to code](#)

Base support interface for NiftySeg commands.

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a 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
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
```

Outputs:

```
None
```

72.2 interfaces.niftyseg.em

72.2.1 EM

[Link to code](#)

Wraps the executable 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 -bc_order 3 -bc_thresh 0 -max_iter 100 -min_iter 0 -nopriors_
↪4 -bc_out im1_bc_em.nii.gz -out im1_em.nii.gz -out_outlier im1_outlier_em.nii.gz
↪'
```

Inputs:

```
[Mandatory]
no_prior: (an integer (int or long))
    Number of classes to use without prior
    argument: ``-nopriors %s``
    mutually_exclusive: prior_4D, priors
in_file: (an existing file name)
    Input image to segment
    argument: ``-in %s``, position: 4
priors: (a list of items which are any value)
    List of priors filepaths.
    argument: ``%s``
    mutually_exclusive: no_prior, prior_4D
prior_4D: (an existing file name)
    4D file containing the priors
    argument: ``-prior4D %s``
    mutually_exclusive: no_prior, priors

[Optional]
max_iter: (an integer (int or long), nipy default value: 100)
    Maximum number of iterations
    argument: ``-max_iter %s``
out_bc_file: (a file name)
    Output bias corrected image
    argument: ``-bc_out %s``
mrf_beta_val: (a float)
    Weight of the Markov Random Field
    argument: ``-mrf_beta %s``
out_file: (a file name)
    Output segmentation
    argument: ``-out %s``
min_iter: (an integer (int or long), nipy default value: 0)
    Minimum number of iterations
    argument: ``-min_iter %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/
    argument: ``-rf %s %s``
bc_order_val: (an integer (int or long), nipy default value: 3)
    Polynomial order for the bias field
    argument: ``-bc_order %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
bc_thresh_val: (a float, nipy default value: 0)
```

(continues on next page)

(continued from previous page)

```

        Bias field correction will run only if the ratio of improvement is
        below bc_thresh. (default=0 [OFF])
        argument: ``-bc_thresh %s``
mask_file: (an existing file name)
        Filename of the ROI for label fusion
        argument: ``-mask %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%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]
        argument: ``-outlier %s %s``
out_outlier_file: (a file name)
        Output outlieriness image
        argument: ``-out_outlier %s``
reg_val: (a float)
        Amount of regularization over the diagonal of the covariance matrix
        [above 1]
        argument: ``-reg %s``

```

Outputs:

```

out_outlier_file: (a file name)
        Output outlieriness image
out_bc_file: (a file name)
        Output bias corrected image
out_file: (a file name)
        Output segmentation

```

72.3 interfaces.niftyseg.label_fusion

72.3.1 CalcTopNCC

[Link to code](#)Wraps the executable 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]
top_templates: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

    Number of Top Templates
    argument: ``-n %s``, position: 4
in_templates: (a list of items which are an existing file name)
    argument: ``%s``, position: 3
in_file: (an existing file name)
    Target file
    argument: ``-target %s``, position: 1
num_templates: (an integer (int or long))
    Number of Templates
    argument: ``-templates %s``, position: 2

[Optional]
mask_file: (an existing file name)
    Filename of the ROI for label fusion
    argument: ``-mask %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```
out_files: (any value)
```

72.3.2 LabelFusion[Link to code](#)

Wraps the executable 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'

```

(continues on next page)

(continued from previous page)

```
>>> 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.
    argument: ``-s``, position: 2
in_file: (an existing file name)
    Filename of the 4D integer label image.
    argument: ``-in %s``, position: 1
file_to_seg: (an existing file name)
    Original image to segment (3D Image)

[Optional]
verbose: ('0' or '1' or '2')
    Verbose level [0 = off, 1 = on, 2 = debug] (default = 0)
    argument: ``-v %s``
unc_thresh: (a float)
    If <float> percent of labels agree, then area is not uncertain.
    argument: ``-unc thresh %f``
template_num: (an integer (int or long))
    Number of labels to use
template_file: (an existing file name)
    Registered templates (4D 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
mrf_value: (a float)
    MRF prior strength (between 0 and 5)
    argument: ``-MRF_beta %f``
proportion: (a float)
    Proportion of the label (only for single labels).
    argument: ``-prop %s``
out_file: (a file name)
    Output consensus segmentation
    argument: ``-out %s``
max_iter: (an integer (int or long))
    Maximum number of iterations (default = 15).
    argument: ``-max_iter %d``
prob_update_flag: (a boolean)
    Update label proportions at each iteration
    argument: ``-prop_update``
sm_ranking: ('ALL' or 'GNCC' or 'ROINCC' or 'LNCC', nipy default
    value: ALL)
    Ranking for STAPLE and MV
    argument: ``-s``, position: 3
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)
    argument: ``-setPQ %f %f``
unc: (a boolean)
    Only consider non-consensus voxels to calculate statistics
    argument: ``-unc``
mask_file: (an existing file name)
    Filename of the ROI for label fusion
```

(continues on next page)

(continued from previous page)

```

        argument: ``-mask %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
kernel_size: (a float)
    Gaussian kernel size in mm to compute the local similarity
dilation_roi: (an integer (int or long))
    Dilation of the ROI ( <int> d>=1 )
conv: (a float)
    Ratio for convergence (default epsilon = 10^-5).
    argument: ``-conv %f``
prob_flag: (a boolean)
    Probabilistic/Fuzzy segmented image
    argument: ``-outProb``

```

Outputs:

```

out_file: (an existing file name)
    image written after calculations

```

72.4 interfaces.niftyseg.lesions

72.4.1 FillLesions

[Link to code](#)Wraps the executable 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 nipyseg.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]
lesion_mask: (an existing file name)
    Lesion mask
    argument: ``-l %s``, position: 2
in_file: (an existing file name)
    Input image to fill lesions
    argument: ``-i %s``, position: 1

[Optional]
search: (a float)
    Minimum percentage of valid voxels in target patch <float> (by
    default 0).
    argument: ``-search %f``
out_file: (a file name)

```

(continues on next page)

(continued from previous page)

```

    The output filename of the fill lesions results
    argument: ``-o %s``, position: 3
out_datatype: (a string)
    Set output <datatype> (char, short, int, uchar, ushort, uint, float,
    double).
    argument: ``-odt %s``
cwf: (a float)
    Patch cardinality weighting factor (by default 2).
    argument: ``-cwf %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
size: (an integer (int or long))
    Search regions size respect biggest patch size (by default 4).
    argument: ``-size %d``
match: (a float)
    Percentage of minimum number of voxels between patches <float> (by
    default 0.5).
    argument: ``-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).
    argument: ``-other``
use_2d: (a boolean)
    Uses 2D patches in the Z axis, by default 3D.
    argument: ``-2D``
in_dilation: (an integer (int or long))
    Dilate the mask <int> times (in voxels, by default 0)
    argument: ``-dil %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
smooth: (a float)
    Smoothing by <float> (in minimal 6-neighbourhood voxels (by default
    0.1)).
    argument: ``-smo %f``
verbose: (a boolean)
    Verbose (by default OFF).
    argument: ``-v``
debug: (a boolean)
    Save all intermidium files (by default OFF).
    argument: ``-debug``
bin_mask: (a file name)
    Give a binary mask with the valid search areas.
    argument: ``-mask %s``

```

Outputs:

```

out_file: (a file name)
    Output segmentation

```

72.5 interfaces.niftyseg.maths

72.5.1 BinaryMaths

[Link to code](#)

Wraps the executable 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
```

(continues on next page)

(continued from previous page)

```

>>> 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]
operand_file: (an existing file name)
    second image to perform operation with
    argument: ``%s``, position: 5
    mutually_exclusive: operand_value, operand_str
in_file: (an existing file name)
    image to operate on
    argument: ``%s``, position: 2
operand_str: ('x' or 'y' or 'z')
    string value to perform operation splitinter
    argument: ``%s``, position: 5
    mutually_exclusive: operand_value, operand_file
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
    argument: ``-%s``, position: 4
operand_value: (a float)
    float value to perform operation with
    argument: ``%.8f``, position: 5
    mutually_exclusive: operand_file, operand_str

[Optional]
out_file: (a file name)
    image to write
    argument: ``%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)
    argument: ``-odt %s``, position: -3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```
out_file: (a file name)
          image written after calculations
```

72.5.2 BinaryMathsInteger

[Link to code](#)

Wraps the executable 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
         argument: ``%s``, position: 2
operation: ('dil' or 'ero' or 'tp' or 'equal' or 'pad' or 'crop')
```

(continues on next page)

(continued from previous page)

```

        operation to perform
        argument: ``-s``, position: 4
operand_value: (an integer (int or long))
        int value to perform operation with
        argument: ``%d``, position: 5

[Optional]
out_file: (a file name)
        image to write
        argument: ``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)
        argument: ``-odt %s``, position: -3
args: (a unicode string)
        Additional parameters to the command
        argument: ``s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables

```

Outputs:

```

out_file: (a file name)
        image written after calculations

```

72.5.3 MathsCommand[Link to code](#)

Wraps the executable 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
        argument: ``s``, position: 2

[Optional]
args: (a unicode string)
        Additional parameters to the command
        argument: ``s``
out_file: (a file name)
        image to write
        argument: ``s``, position: -2
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double'
        or 'input')

```

(continues on next page)

(continued from previous page)

```

    datatype to use for output (default uses input type)
    argument: ``-odt %s``, 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

```

Outputs:

```

out_file: (a file name)
          image written after calculations

```

72.5.4 Merge[Link to code](#)Wraps the executable 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 nipy.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]
in_file: (an existing file name)
         image to operate on
         argument: ``%s``, position: 2
dimension: (an integer (int or long))
           Dimension to merge the images.
merge_files: (a list of items which are an existing file name)
             List of images to merge to the working image <input>.
             argument: ``%s``, position: 4

[Optional]
out_file: (a file name)
         image to write
         argument: ``%s``, position: -2
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
output_datatype: ('float' or 'char' or 'int' or 'short' or 'double')

```

(continues on next page)

(continued from previous page)

```

    or 'input')
    datatype to use for output (default uses input type)
    argument: ``-odt %s``, 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

```

Outputs:

```

out_file: (a file name)
    image written after calculations

```

72.5.5 TupleMaths[Link to code](#)Wraps the executable 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 nipy.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)

```

(continues on next page)

(continued from previous page)

```

>>> 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]
operand_file2: (an existing file name)
    image to perform operation 2 with
    argument: ``%s``, position: 6
    mutually_exclusive: operand_value2
operand_file1: (an existing file name)
    image to perform operation 1 with
    argument: ``%s``, position: 5
    mutually_exclusive: operand_value1
operand_value1: (a float)
    float value to perform operation 1 with
    argument: ``%.8f``, position: 5
    mutually_exclusive: operand_file1
operation: ('lncc' or 'lssd' or 'lltsnorm')
    operation to perform
    argument: ``-%s``, position: 4
in_file: (an existing file name)
    image to operate on
    argument: ``%s``, position: 2
operand_value2: (a float)
    float value to perform operation 2 with
    argument: ``%.8f``, position: 6
    mutually_exclusive: operand_file2

[Optional]
out_file: (a file name)
    image to write
    argument: ``%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)
    argument: ``-odt %s``, position: -3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

out_file: (a file name)
    image written after calculations

```

72.5.6 UnaryMaths[Link to code](#)Wraps the executable 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
```

(continues on next page)

(continued from previous page)

```

>>> 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
        argument: ``%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
        argument: ``-%s``, position: 4

[Optional]
out_file: (a file name)
        image to write
        argument: ``%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)
        argument: ``-odt %s``, position: -3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
environ: (a dictionary with keys which are a 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

```

Outputs:

```

out_file: (a file name)
        image written after calculations

```

72.6 interfaces.niftyseg.patchmatch

72.6.1 PatchMatch

[Link to code](#)Wraps the executable 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]
in_file: (an existing file name)
    Input image to segment
    argument: ``-i %s``, position: 1
mask_file: (an existing file name)
    Input mask for the area where applies PatchMatch
    argument: ``-m %s``, position: 2
database_file: (an existing file name)
    Database with the segmentations
    argument: ``-db %s``, position: 3

[Optional]
it_num: (an integer (int or long))
    Number of iterations for the patchmatch algorithm
    argument: ``-it %i``
match_num: (an integer (int or long))
    Number of better matching
    argument: ``-match %i``
cs_size: (an integer (int or long))
    Constrained search area size, number of times bigger than the
    patchsize
    argument: ``-cs %i``
out_file: (a file name)
    The output filename of the patchmatch results
    argument: ``-o %s``, position: 4
patch_size: (an integer (int or long))
    Patch size, #voxels
    argument: ``-size %i``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
pm_num: (an integer (int or long))
    Number of patchmatch executions
    argument: ``-pm %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
```

Outputs:

```
out_file: (a file name)
          Output segmentation
```

72.7 interfaces.niftyseg.stats

72.7.1 BinaryStats

[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: 2
operand_file: (an existing file name)
    second image to perform operation with
    argument: ``%s``, position: 5
    mutually_exclusive: operand_value
operation: ('p' or 'sa' or 'ss' or 'svp' or 'al' or 'd' or 'ncc' or
    'nmi' or 'Vl' or 'Nl')
    operation to perform
    argument: ``-%s``, position: 4
operand_value: (a float)
    value to perform operation with
    argument: ``%.8f``, position: 5
    mutually_exclusive: operand_file

[Optional]
mask_file: (an existing file name)
    statistics within the masked area
    argument: ``-m %s``, position: -2
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
larger_voxel: (a float)
    Only estimate statistics if voxel is larger than <float>
    argument: ``-t %f``, 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
```

Outputs:

```
output: (an array)
    Output array from seg_stats
```

72.7.2 StatsCommand

[Link to code](#)

Wraps the executable 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
    argument: ``%s``, position: 2

[Optional]
larger_voxel: (a float)
```

(continues on next page)

(continued from previous page)

```

    Only estimate statistics if voxel is larger than <float>
    argument: ``-t %f``, position: -3
mask_file: (an existing file name)
    statistics within the masked area
    argument: ``-m %s``, position: -2
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

output: (an array)
    Output array from seg_stats

```

72.7.3 UnaryStats

[Link to code](#)Wraps the executable 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.

xdim - Output the voxel dimation 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'

```

(continues on next page)

(continued from previous page)

```

>>> unary_v.cmdline
'seg_stats iml.nii -v'
>>> unary_v.run()
>>> # Test vl operation
>>> unary_vl = copy.deepcopy(unary)
>>> unary_vl.inputs.operation = 'vl'
>>> unary_vl.cmdline
'seg_stats iml.nii -vl'
>>> unary_vl.run()
>>> # Test x operation
>>> unary_x = copy.deepcopy(unary)
>>> unary_x.inputs.operation = 'x'
>>> unary_x.cmdline
'seg_stats iml.nii -x'
>>> unary_x.run()

```

Inputs:

```

[Mandatory]
in_file: (an existing file name)
        image to operate on
        argument: ``%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
        argument: ``-%s``, position: 4

[Optional]
mask_file: (an existing file name)
           statistics within the masked area
           argument: ``-m %s``, position: -2
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
larger_voxel: (a float)
              Only estimate statistics if voxel is larger than <float>
              argument: ``-t %f``, 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

```

Outputs:

```

output: (an array)
        Output array from seg_stats

```


73.1 interfaces.nipy.base

73.1.1 NipyBaseInterface

[Link to code](#)

Inputs:

None

Outputs:

None

73.2 interfaces.nipy.model

73.2.1 EstimateContrast

[Link to code](#)

Estimate contrast of a fitted model.

Inputs:

```
[Mandatory]
reg_names: (a list of items which are any value)
axis: (any value)
nvbeta: (any value)
dof: (any value)
        degrees of freedom
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
```

(continues on next page)

(continued from previous page)

```

    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.
constants: (any value)
s2: (an existing file name)
    squared variance of the residuals
beta: (an existing file name)
    beta coefficients of the fitted model

[Optional]
mask: (a file name)

```

Outputs:

```

stat_maps: (a list of items which are an existing file name)
p_maps: (a list of items which are an existing file name)
z_maps: (a list of items which are an existing file name)

```

73.2.2 FitGLM

[Link to code](#)

Fit GLM model based on the specified design. Supports only single or concatenated runs.

Inputs:

```

[Mandatory]
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)
TR: (a float)

[Optional]
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
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'
save_residuals: (a boolean, nipyne default value: False)
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'
normalize_design_matrix: (a boolean, nipyne default value: False)
    normalize (zscore) the regressors before fitting
plot_design_matrix: (a boolean, nipyne default value: False)

```

Outputs:

```
axis: (any value)
nvbeta: (any value)
constants: (any value)
a: (an existing file name)
dof: (any value)
beta: (an existing file name)
s2: (an existing file name)
reg_names: (a list of items which are any value)
residuals: (a file name)
```

73.3 interfaces.nipy.preprocess

73.3.1 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
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.
cc: (a boolean)
    Keep only the largest connected component
```

Outputs:

```
brain_mask: (an existing file name)
```

73.3.2 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 nipy.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]
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.
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
tr: (a float)
    TR in seconds
    requires: slice_times
```

Outputs:

```
par_file: (a list of items which are an existing file name)
          Motion parameter files. Angles are not euler angles
out_file: (a list of items which are an existing file name)
          Realigned files
```

73.3.3 Trim

[Link to code](#)

Simple interface to trim a few volumes from a 4d fmri nifti file

Examples

```
>>> from nipy.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]
suffix: (a unicode string, nipy default value: _trim)
        suffix for out_file to use if no out_file provided
end_index: (an integer (int or long), nipy default value: 0)
        last volume indexed as in python (and 0 for last)
begin_index: (an integer (int or long), nipy default value: 0)
        first volume
out_file: (a file name)
        output filename
```

Outputs:

```
out_file: (an existing file name)
```

73.4 interfaces.nipy.utils

73.4.1 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 `nipy.algorithms.metrics.Similarity` instead.

Example

```
>>> from nipy.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]
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
```

(continues on next page)

(continued from previous page)

```
        histogram as an input and return a float.  
mask2: (an existing file name)  
        3D volume  
mask1: (an existing file name)  
        3D volume
```

Outputs:

```
similarity: (a float)  
            Similarity between volume 1 and 2
```

74.1 interfaces.nitime.analysis

74.1.1 CoherenceAnalyzer

[Link to code](#)

Inputs:

```
[Optional]
in_TS: (any value)
    a nitime TimeSeries object
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,...
output_csv_file: (a file name)
    File to write outputs (coherence,time-delay) with file-names:
    file_name_{coherence,timedelay}
n_overlap: (a long integer >= 0, nipype default value: 0)
    The number of samples which overlapbetween subsequent
    windows.(Defaults to 0)
NFFT: (a long integer >= 32, nipype 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.)
figure_type: ('matrix' or 'network', nipype default value: matrix)
    The type of plot to generate, where 'matrix' denotes a matrix image
    and'network' denotes a graph representation. Default: 'matrix'
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
frequency_range: (a list of from 2 to 2 items which are any value,
    nipype default value: [0.02, 0.15])
    The range of frequencies overwhich the analysis will
    average.[low,high] (Default [0.02,0.15]
TR: (a float)
    The TR used to collect the datain your csv file <in_file>
```

Outputs:

```
coherence_array: (an array)
    The pairwise coherence valuesbetween the ROIs
timedelay_array: (an array)
    The pairwise time delays between theROIs (in seconds)
timedelay_fig: (a file name)
    Figure representing coherence values
coherence_fig: (a file name)
    Figure representing coherence values
coherence_csv: (a file name)
    A csv file containing the pairwise coherence values
timedelay_csv: (a file name)
    A csv file containing the pairwise time delay values
```

74.2 interfaces.nitime.base

74.2.1 NitimeBaseInterface

[Link to code](#)

Inputs:

```
None
```

Outputs:

```
None
```

75.1 interfaces.semtools.brains.classify

75.1.1 BRAINSPosteriorToContinuousClass

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputCsfVolume: (an existing file name)
    CSF Posterior Volume
    argument: ``--inputCsfVolume %s``
inputWhiteVolume: (an existing file name)
    White Matter Posterior Volume
    argument: ``--inputWhiteVolume %s``
inputCrblWmVolume: (an existing file name)
    Cerebellum White Matter Posterior Volume
    argument: ``--inputCrblWmVolume %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputSurfaceGmVolume: (an existing file name)
    Surface Grey Matter Posterior Volume
    argument: ``--inputSurfaceGmVolume %s``
inputCrblGmVolume: (an existing file name)
    Cerebellum Grey Matter Posterior Volume
```

(continues on next page)

(continued from previous page)

```

    argument: ``--inputCrb1GmVolume %s``
inputVbVolume: (an existing file name)
    Venous Blood Posterior Volume
    argument: ``--inputVbVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Output Continuous Tissue Classified Image
    argument: ``--outputVolume %s``
inputBasalGmVolume: (an existing file name)
    Basal Grey Matter Posterior Volume
    argument: ``--inputBasalGmVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Output Continuous Tissue Classified Image

```

75.2 interfaces.semtools.brains.segmentation

75.2.1 BRAINSTalairach

[Link to code](#)

Wraps the executable 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:

```

[Optional]
IRPisIndex: (a boolean)
    IRP Point is Index
    argument: ``--IRPisIndex ``
PCisIndex: (a boolean)
    PC Point is Index
    argument: ``--PCisIndex ``
inputVolume: (an existing file name)
    Input image used to define physical space of images
    argument: ``--inputVolume %s``
SLAisIndex: (a boolean)
    SLA Point is Index
    argument: ``--SLAisIndex ``

```

(continues on next page)

(continued from previous page)

```

IRP: (a list of items which are a float)
      Location of IRP Point
      argument: ``--IRP %s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
          Environment variables
outputGrid: (a boolean or a file name)
            Name of the resulting Talairach Grid file
            argument: ``--outputGrid %s``
SLA: (a list of items which are a float)
      Location of SLA Point
      argument: ``--SLA %s``
outputBox: (a boolean or a file name)
            Name of the resulting Talairach Bounding Box file
            argument: ``--outputBox %s``
args: (a unicode string)
       Additional parameters to the command
       argument: ``%s``
ACisIndex: (a boolean)
            AC Point is Index
            argument: ``--ACisIndex ``
AC: (a list of items which are a float)
     Location of AC Point
     argument: ``--AC %s``
PC: (a list of items which are a float)
     Location of PC Point
     argument: ``--PC %s``

```

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

```

75.2.2 BRAINSTalairachMask[Link to code](#)

Wraps the executable 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:

```

[Optional]
hemisphereMode: ('left' or 'right' or 'both')

```

(continues on next page)

(continued from previous page)

```

    Mode for box creation: left, right, both
    argument: ``--hemisphereMode %s``
talairachBox: (an existing file name)
    Name of the Talairach box file.
    argument: ``--talairachBox %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
expand: (a boolean)
    Expand exterior box to include surface CSF
    argument: ``--expand``
inputVolume: (an existing file name)
    Input image used to define physical space of resulting mask
    argument: ``--inputVolume %s``
talairachParameters: (an existing file name)
    Name of the Talairach parameter file.
    argument: ``--talairachParameters %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Output filename for the resulting binary image
    argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Output filename for the resulting binary image

```

75.2.3 SimilarityIndex[Link to code](#)

Wraps the executable 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: Eunyoung Regin Kim

Inputs:

```

[Optional]
thresholdInterval: (a float)
    Threshold interval to compute similarity index between zero and one
    argument: ``--thresholdInterval %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputManualVolume: (an existing file name)
    input manual(reference) volume
    argument: ``--inputManualVolume %s``
ANNContinuousVolume: (an existing file name)
    ANN Continuous volume to be compared to the manual volume
    argument: ``--ANNContinuousVolume %s``
outputCSVFilename: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    output CSV Filename
    argument: ``--outputCSVFilename %s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
Environment variables

```

Outputs:

None

75.3 interfaces.semtools.brains.utilities

75.3.1 GenerateEdgeMapImage

[Link to code](#)

Wraps the executable command “GenerateEdgeMapImage”.

title: GenerateEdgeMapImage

category: BRAINS.Utilities

description: Automatic edgemap generation for edge-guided super-resolution reconstruction

version: 1.0

contributor: Ali Ghayoor

Inputs:

```

[Optional]
lowerPercentileMatching: (a float)
    Map lower quantile and below to minOutputRange. It should be a value
    between zero and one
    argument: ``--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
    argument: ``--maximumOutputRange %d``
upperPercentileMatching: (a float)
    Map upper quantile and above to maxOutputRange. It should be a value
    between zero and one
    argument: ``--upperPercentileMatching %f``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outputMaximumGradientImage: (a boolean or a file name)
    output gradient image file name
    argument: ``--outputMaximumGradientImage %s``
outputEdgeMap: (a boolean or a file name)
    output edgemap file name
    argument: ``--outputEdgeMap %s``
inputMask: (an existing file name)
    Input mask file name. If set, image histogram percentiles will be
    calculated within the mask
    argument: ``--inputMask %s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a

```

(continues on next page)

(continued from previous page)

```

    value of class 'str', nipy default value: {})
    Environment variables
inputMRVolumes: (a list of items which are an existing file name)
    List of input structural MR volumes to create the maximum edgemap
    argument: ``--inputMRVolumes %s...``
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
    argument: ``--minimumOutputRange %d``

```

Outputs:

```

outputMaximumGradientImage: (an existing file name)
    output gradient image file name
outputEdgeMap: (an existing file name)
    (required) output file name

```

75.3.2 GeneratePurePlugMask[Link to code](#)

Wraps the executable 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:

```

[Optional]
inputImageModalities: (a list of items which are an existing file
    name)
    List of input image file names to create pure plugs mask
    argument: ``--inputImageModalities %s...``
outputMaskFile: (a boolean or a file name)
    Output binary mask file name
    argument: ``--outputMaskFile %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
threshold: (a float)
    threshold value to define class membership
    argument: ``--threshold %f``
numberOfSubSamples: (a list of items which are an integer (int or
    long))
    Number of continuous index samples taken at each direction of lattice
    space for each plug volume
    argument: ``--numberOfSubSamples %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

outputMaskFile: (an existing file name)
    (required) Output binary mask file name

```

75.3.3 HistogramMatchingFilter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
numberOfHistogramBins: (an integer (int or long))
    number of histogram bin
    argument: ``--numberOfHistogramBins %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputBinaryVolume: (an existing file name)
    inputBinaryVolume
    argument: ``--inputBinaryVolume %s``
inputVolume: (an existing file name)
    The Input image to be computed for statistics
    argument: ``--inputVolume %s``
referenceVolume: (an existing file name)
    The Input image to be computed for statistics
    argument: ``--referenceVolume %s``
histogramAlgorithm: ('OtsuHistogramMatching')
    histogram algrithm selection
    argument: ``--histogramAlgorithm %s``
numberOfMatchPoints: (an integer (int or long))
    number of histogram matching points
    argument: ``--numberOfMatchPoints %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
verbose: (a boolean)
    verbose mode running for debbuging
    argument: ``--verbose ``
referenceBinaryVolume: (an existing file name)
    referenceBinaryVolume
    argument: ``--referenceBinaryVolume %s``
writeHistogram: (a unicode string)
    decide if histogram data would be written with prefixe of the file
    name
    argument: ``--writeHistogram %s``
outputVolume: (a boolean or a file name)
    Output Image File Name
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Output Image File Name
```

75.4 interfaces.semtools.converters

75.4.1 DWICompare

[Link to code](#)

Wraps the executable 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 identical. 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:

```
[Optional]
inputVolume2: (an existing file name)
    Second input volume (.nhdr or .nrrd)
    argument: ``--inputVolume2 %s``
inputVolume1: (an existing file name)
    First input volume (.nhdr or .nrrd)
    argument: ``--inputVolume1 %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

None

75.4.2 DWISimpleCompare

[Link to code](#)

Wraps the executable 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 identical. 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:

```
[Optional]
inputVolume2: (an existing file name)
    Second input volume (.nhdr or .nrrd)
    argument: ``--inputVolume2 %s``
inputVolume1: (an existing file name)
    First input volume (.nhdr or .nrrd)
    argument: ``--inputVolume1 %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
checkDWIData: (a boolean)
    check for existence of DWI data, and if present, compare it
    argument: ``--checkDWIData ``
```

Outputs:

None

75.5 interfaces.semtools.diffusion.diffusion

75.5.1 DWIConvert

[Link to code](#)

Wraps the executable 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:

```
[Optional]
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.
    argument: ``--writeProtocolGradientsFile ``
```

(continues on next page)

(continued from previous page)

```

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 empirically computed from the private
    BMatrix fields. In some cases the private BMatrix is consistent with
    the public gradients, but not in all cases, when it exists BMatrix
    is usually most robust.
    argument: ``--useBMatrixGradientDirections``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--conversionMode %s``
inputVolume: (an existing file name)
    Input DWI volume -- not used for DicomToNrrd mode.
    argument: ``--inputVolume %s``
outputVolume: (a boolean or a file name)
    Output filename (.nhdr or .nrrd)
    argument: ``--outputVolume %s``
outputBVectors: (a boolean or a file name)
    The Gradient Vectors are stored in FSL .bvec text file format
    (defaults to <outputVolume>.bvec)
    argument: ``--outputBVectors %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
inputBVectors: (an existing file name)
    The Gradient Vectors are stored in FSL .bvec text file format
    argument: ``--inputBVectors %s``
fMRI: (a boolean)
    Output a NRRD file, but without gradients
    argument: ``--fMRI``
inputBValues: (an existing file name)
    The B Values are stored in FSL .bval text file format
    argument: ``--inputBValues %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.
    argument: ``--smallGradientThreshold %f``
outputDirectory: (a boolean or a directory name)
    Directory holding the output NRRD file
    argument: ``--outputDirectory %s``
outputBValues: (a boolean or a file name)
    The B Values are stored in FSL .bval text file format (defaults to
    <outputVolume>.bval)
    argument: ``--outputBValues %s``
allowLossyConversion: (a boolean)

```

(continues on next page)

(continued from previous page)

```

        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!
        argument: ``--allowLossyConversion``
inputDicomDirectory: (an existing directory name)
        Directory holding Dicom series
        argument: ``--inputDicomDirectory %s``
useIdentityMeaseurementFrame: (a boolean)
        Adjust all the gradients so that the measurement frame is an
        identity matrix.
        argument: ``--useIdentityMeaseurementFrame``
gradientVectorFile: (a boolean or a file name)
        Text file giving gradient vectors
        argument: ``--gradientVectorFile %s``
fslNIFTIFile: (an existing file name)
        4D NIFTI file containing gradient volumes
        argument: ``--fslNIFTIFile %s``
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.
        argument: ``--transposeInputBVectors``

```

Outputs:

```

outputDirectory: (an existing directory name)
        Directory holding the output NRRD file
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)
gradientVectorFile: (an existing file name)
        Text file giving gradient vectors
outputVolume: (an existing file name)
        Output filename (.nhdr or .nrrd)

```

75.5.2 dtiaverage[Link to code](#)

Wraps the executable command “ dtiaverage “.

title: DTIAverage (DTIPProcess)

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 aftSeveral 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/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:

```
[Optional]
tensor_output: (a boolean or a file name)
    Averaged tensor volume
    argument: ``--tensor_output %s``
DTI_double: (a boolean)
    Tensor components are saved as doubles (cannot be visualized in
    Slicer)
    argument: ``--DTI_double ``
inputs: (a list of items which are an existing file name)
    List of all the tensor fields to be averaged
    argument: ``--inputs %s...``
verbose: (a boolean)
    produce verbose output
    argument: ``--verbose ``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
tensor_output: (an existing file name)
    Averaged tensor volume
```

75.5.3 dtiestim[Link to code](#)

Wraps the executable 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 lls|wls|nls|ml` . 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:

```
[Optional]
step: (a float)
    Gradient descent step size (for nls and ml methods)
    argument: ``--step %f``
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
    argument: ``--B0_mask_output %s``
correction: ('none' or 'zero' or 'abs' or 'nearest')
    Correct the tensors if computed tensor is not semi-definite positive
    argument: ``--correction %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 stictly positive
    eigenvalues
    argument: ``--shiftNeg ``
bad_region_mask: (an existing file name)
    Bad region mask. Image where for every voxel > 0 the tensors are not
    estimated
    argument: ``--bad_region_mask %s``
sigma: (a float)
    argument: ``--sigma %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
dwi_image: (an existing file name)
    DWI image volume (required)
    argument: ``--dwi_image %s``
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)
    argument: ``--method %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
```

(continues on next page)

(continued from previous page)

```

    NOT desired, then use option -t 0.
    argument: ``--brain_mask %s``
idwi: (a boolean or a file name)
    idwi output image. Image with isotropic diffusion-weighted
    information = geometric mean of diffusion images
    argument: ``--idwi %s``
DTI_double: (a boolean)
    Tensor components are saved as doubles (cannot be visualized in
    Slicer)
    argument: ``--DTI_double ``
threshold: (an integer (int or long))
    Baseline threshold for estimation. If not specified calculated using
    an OTSU threshold on the baseline image.
    argument: ``--threshold %d``
B0: (a boolean or a file name)
    Baseline image, average of all baseline images
    argument: ``--B0 %s``
verbose: (a boolean)
    produce verbose output
    argument: ``--verbose ``
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).
    argument: ``--shiftNegCoeff %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
tensor_output: (a boolean or a file name)
    Tensor OutputImage
    argument: ``--tensor_output %s``
weight_iterations: (an integer (int or long))
    Number of iterations to recalculate weightings from tensor estimate
    argument: ``--weight_iterations %d``
defaultTensor: (a list of items which are a float)
    Default tensor used if estimated tensor is below a given threshold
    argument: ``--defaultTensor %s``

```

Outputs:

```

B0: (an existing file name)
    Baseline image, average of all baseline images
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
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

```

75.5.4 dtiprocess[Link to code](#)

Wraps the executable command “ dtiprocess “.

title: DTIProcess (DTIProcess)

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 mynewdofile.dof >> mynewdofile.txt dtiprocess -dti_image mytensorfield.nhdr -newdof_file mynewdofile.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:

```
[Optional]
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
    argument: ``--mask %s``
newdof_file: (an existing file name)
    Transformation file for affine transformation. RView NEW format.
    (txt file output of dof2mat)
    argument: ``--newdof_file %s``
correction: ('none' or 'zero' or 'abs' or 'nearest')
    Correct the tensors if computed tensor is not semi-definite positive
    argument: ``--correction %s``
fa_gradient_output: (a boolean or a file name)
    Fractional Anisotropy Gradient output file
    argument: ``--fa_gradient_output %s``
RD_output: (a boolean or a file name)
    RD (Radial Diffusivity 1/2*(lambda2+lambda3)) output
```

(continues on next page)

(continued from previous page)

```

        argument: ``--RD_output %s``
affineitk_file: (an existing file name)
    Transformation file for affine transformation. ITK format.
    argument: ``--affineitk_file %s``
frobenius_norm_output: (a boolean or a file name)
    Frobenius Norm Output
    argument: ``--frobenius_norm_output %s``
lambda2_output: (a boolean or a file name)
    Lambda 2 (middle eigenvalue) output
    argument: ``--lambda2_output %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
DTI_double: (a boolean)
    Tensor components are saved as doubles (cannot be visualized in
    Slicer)
    argument: ``--DTI_double ``
outmask: (a boolean or a file name)
    Name of the masked tensor field.
    argument: ``--outmask %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.
    argument: ``--negative_eigenvector_output %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.
    argument: ``--deformation_output %s``
dof_file: (an existing file name)
    Transformation file for affine transformation. This can be ITK
    format (or the outdated RView).
    argument: ``--dof_file %s``
reorientation: ('fs' or 'ppd')
    Reorientation type (fs, ppd)
    argument: ``--reorientation %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].
    argument: ``--scalar_float ``
lambda3_output: (a boolean or a file name)
    Lambda 3 (smallest eigenvalue) output
    argument: ``--lambda3_output %s``
md_output: (a boolean or a file name)
    Mean Diffusivity output file
    argument: ``--md_output %s``
interpolation: ('nearestneighbor' or 'linear' or 'cubic')
    Interpolation type (nearestneighbor, linear, cubic)
    argument: ``--interpolation %s``
lambda1_output: (a boolean or a file name)
    Axial Diffusivity - Lambda 1 (largest eigenvalue) output
    argument: ``--lambda1_output %s``
principal_eigenvector_output: (a boolean or a file name)
    Principal Eigenvectors Output
    argument: ``--principal_eigenvector_output %s``
fa_gradmag_output: (a boolean or a file name)
    Fractional Anisotropy Gradient Magnitude output file

```

(continues on next page)

(continued from previous page)

```

        argument: ``--fa_gradmag_output %s``
color_fa_output: (a boolean or a file name)
    Color Fractional Anisotropy output file
    argument: ``--color_fa_output %s``
sigma: (a float)
    Scale of gradients
    argument: ``--sigma %f``
fa_output: (a boolean or a file name)
    Fractional Anisotropy output file
    argument: ``--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.
    argument: ``--forward %s``
hField: (a boolean)
    forward and inverse transformations are h-fields instead of
    displacement fields
    argument: ``--hField ``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    produce verbose output
    argument: ``--verbose ``
dti_image: (an existing file name)
    DTI tensor volume
    argument: ``--dti_image %s``
rot_output: (a boolean or a file name)
    Rotated tensor output file. Must also specify the dof file.
    argument: ``--rot_output %s``

```

Outputs:

```

lambda3_output: (an existing file name)
    Lambda 3 (smallest eigenvalue) output
lambda1_output: (an existing file name)
    Axial Diffusivity - Lambda 1 (largest eigenvalue) output
fa_gradient_output: (an existing file name)
    Fractional Anisotropy Gradient output file
RD_output: (an existing file name)
    RD (Radial Diffusivity  $1/2 * (\lambda_2 + \lambda_3)$ ) output
frobenius_norm_output: (an existing file name)
    Frobenius Norm Output
principal_eigenvector_output: (an existing file name)
    Principal Eigenvectors Output
color_fa_output: (an existing file name)
    Color Fractional Anisotropy output file
lambda2_output: (an existing file name)
    Lambda 2 (middle eigenvalue) output
outmask: (an existing file name)
    Name of the masked tensor field.
fa_output: (an existing file name)
    Fractional Anisotropy 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.
deformation_output: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    Warped tensor field based on a deformation field. This option
    requires the --forward, -F transformation to be specified.
fa_gradmag_output: (an existing file name)
    Fractional Anisotropy Gradient Magnitude output file
md_output: (an existing file name)
    Mean Diffusivity output file
rot_output: (an existing file name)
    Rotated tensor output file. Must also specify the dof file.

```

75.6 interfaces.semtools.diffusion.gtract

75.6.1 compareTractInclusion

[Link to code](#)

Wraps the executable 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:

```

[Optional]
testForFiberCardinality: (a boolean)
    Flag to require the same number of fibers in both tracts
    argument: ``--testForFiberCardinality ``
testForBijection: (a boolean)
    Flag to apply the closeness criterion both ways
    argument: ``--testForBijection ``
writeXMLPolyDataFile: (a boolean)
    Flag to make use of XML files when reading and writing vtkPolyData.
    argument: ``--writeXMLPolyDataFile ``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
standardFiber: (an existing file name)
    Required: standard fiber tract file name
    argument: ``--standardFiber %s``
numberOfPoints: (an integer (int or long))
    Number of points in comparison fiber pairs
    argument: ``--numberOfPoints %d``
testFiber: (an existing file name)
    Required: test fiber tract file name
    argument: ``--testFiber %s``
environ: (a dictionary with keys which are a bytes or None or a value

```

(continues on next page)

(continued from previous page)

```

    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {}))
    Environment variables
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
    argument: ``--closeness %f``

```

Outputs:

None

75.6.2 extractNrrdVectorIndex

[Link to code](#)

Wraps the executable 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:

```

[Optional]
vectorIndex: (an integer (int or long))
    Index in the vector image to extract
    argument: ``--vectorIndex %d``
setImageOrientation: ('AsAcquired' or 'Axial' or 'Coronal' or
    'Sagittal')
    Sets the image orientation of the extracted vector (Axial, Coronal,
    Sagittal)
    argument: ``--setImageOrientation %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Required: input file containing the vector that will be extracted
    argument: ``--inputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the vector image at
    the given index
    argument: ``--outputVolume %s``

```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing the vector image at
    the given index
```

75.6.3 gtractAnisotropyMap

[Link to code](#)

Wraps the executable 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:

```
[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
    argument: ``--anisotropyType %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputTensorVolume: (an existing file name)
    Required: input file containing the diffusion tensor image
    argument: ``--inputTensorVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the selected kind of
    anisotropy scalar.
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing the selected kind of
    anisotropy scalar.
```

75.6.4 gtractAverageBvalues

[Link to code](#)

Wraps the executable 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:

```
[Optional]
directionsTolerance: (a float)
    Tolerance for matching identical gradient direction pairs
    argument: ``--directionsTolerance %f``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
averageB0only: (a boolean)
    Average only baseline gradients. All other gradient directions are
    not averaged, but retained in the outputVolume
    argument: ``--averageB0only ``
inputVolume: (an existing file name)
    Required: input image file name containing multiple baseline
    gradients to average
    argument: ``--inputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing directly averaged
    baseline images
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing directly averaged
    baseline images
```

75.6.5 gtractClipAnisotropy

[Link to code](#)

Wraps the executable 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:

```
[Optional]
clipFirstSlice: (a boolean)
    Clip the first slice of the anisotropy image
    argument: ``--clipFirstSlice ``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Required: input image file name
    argument: ``--inputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the clipped anisotropy
    image
    argument: ``--outputVolume %s``
clipLastSlice: (a boolean)
    Clip the last slice of the anisotropy image
    argument: ``--clipLastSlice ``
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing the clipped anisotropy
    image
```

75.6.6 gtractCoRegAnatomy

[Link to code](#)

Wraps the executable 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:

```

[Optional]
numberOfIterations: (an integer (int or long))
    Number of iterations in the selected 3D fit
    argument: ``--numberOfIterations %d``
numberOfHistogramBins: (an integer (int or long))
    Number of histogram bins
    argument: ``--numberOfHistogramBins %d``
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.
    argument: ``--samplingPercentage %f``
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.
    argument: ``--inputVolume %s``
minimumStepSize: (a float)
    Minimum required step size to move in the selected 3D fit without
    converging -- decrease this to make the fit more exacting
    argument: ``--minimumStepSize %f``
gridSize: (a list of items which are an integer (int or long))
    Number of grid subdivisions in all 3 directions
    argument: ``--gridSize %s``
relaxationFactor: (a float)
    Fraction of gradient from Jacobian to attempt to move in the
    selected 3D fit
    argument: ``--relaxationFactor %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
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.
    argument: ``--useMomentsAlign``
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.
    argument: ``--useGeometryAlign``
gradientTolerance: (a float)
    Gradient Tolerance
    argument: ``--gradientTolerance %f``

```

(continues on next page)

(continued from previous page)

```

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.,
    argument: ``--maxBSplineDisplacement %f``
outputTransformName: (a boolean or a file name)
    Required: filename for the fit transform.
    argument: ``--outputTransformName %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
    argument: ``--translationScale %f``
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.
    argument: ``--numberOfSamples %d``
transformType: ('Rigid' or 'Bspline')
    Transform Type: Rigid|Bspline
    argument: ``--transformType %s``
vectorIndex: (an integer (int or long))
    Vector image index in the moving image (within the DWI) to be used
    for registration.
    argument: ``--vectorIndex %d``
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
    argument: ``--spatialScale %d``
convergence: (a float)
    Convergence Factor
    argument: ``--convergence %f``
maximumStepSize: (a float)
    Maximum permitted step size to move in the selected 3D fit
    argument: ``--maximumStepSize %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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.
    argument: ``--useCenterOfHeadAlign``
borderSize: (an integer (int or long))
    Size of border
    argument: ``--borderSize %d``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.

```

(continues on next page)

(continued from previous page)

```

    argument: ``--numberOfThreads %d``
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.
    argument: ``--inputRigidTransform %s``
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.
    argument: ``--inputAnatomicalVolume %s``

```

Outputs:

```

outputTransformName: (an existing file name)
    Required: filename for the fit transform.

```

75.6.7 gtractConcatDwi

[Link to code](#)

Wraps the executable 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:

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
ignoreOrigins: (a boolean)
    If image origins are different force all images to origin of first
    image
    argument: ``--ignoreOrigins``
inputVolume: (a list of items which are an existing file name)
    Required: input file containing the first diffusion weighted image
    argument: ``--inputVolume %s...``
environ: (a dictionary with keys which are a 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
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the combined diffusion
    weighted images.
    argument: ``--outputVolume %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``

```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD file containing the combined diffusion
    weighted images.
```

75.6.8 gtractCopyImageOrientation

[Link to code](#)

Wraps the executable 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:

```
[Optional]
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Required: input file containing the signed short image to reorient
    without resampling.
    argument: ``--inputVolume %s``
inputReferenceVolume: (an existing file name)
    Required: input file containing orietation that will be cloned.
    argument: ``--inputReferenceVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: name of output NRRD or Nifti file containing the
    reoriented image in reference image space.
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Required: name of output NRRD or Nifti file containing the
    reoriented image in reference image space.
```

75.6.9 gtractCoregBvalues

[Link to code](#)

Wraps the executable 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:

```
[Optional]
numberOfIterations: (an integer (int or long))
    Number of iterations in each 3D fit
    argument: ``--numberOfIterations %d``
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.
    argument: ``--samplingPercentage %f``
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.
    argument: ``--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.
    argument: ``--fixedVolumeIndex %d``
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)
    argument: ``--relaxationFactor %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
    argument: ``--minimumStepSize %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
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
```

(continues on next page)

(continued from previous page)

```

    purposes.
    argument: ``--outputTransform %s``
maximumStepSize: (a float)
    Maximum permitted step size to move in each 3D fit step (adjust when
    eddyCurrentCorrection is enabled; suggested value = 0.1)
    argument: ``--maximumStepSize %f``
debugLevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
    argument: ``--debugLevel %d``
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
    argument: ``--spatialScale %f``
registerB0Only: (a boolean)
    Register the B0 images only
    argument: ``--registerB0Only ``
eddyCurrentCorrection: (a boolean)
    Flag to perform eddy current corection in addition to motion
    correction (recommended)
    argument: ``--eddyCurrentCorrection ``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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.
    argument: ``--movingVolume %s``
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.
    argument: ``--numberOfSpatialSamples %d``
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.
    argument: ``--outputVolume %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``

```

Outputs:

```

outputVolume: (an existing file name)
    Required: name of output NRRD file containing moving images
    individually resampled and fit to the specified fixed image index.
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.

```

75.6.10 gtractCostFastMarching[Link to code](#)

Wraps the executable 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:

```
[Optional]
anisotropyWeight: (a float)
    Anisotropy weight used for vcl_cost function calculations
    argument: ``--anisotropyWeight %f``
seedThreshold: (a float)
    Anisotropy threshold used for seed selection
    argument: ``--seedThreshold %f``
inputAnisotropyVolume: (an existing file name)
    Required: input anisotropy image file name
    argument: ``--inputAnisotropyVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputCostVolume: (a boolean or a file name)
    Output vcl_cost image
    argument: ``--outputCostVolume %s``
outputSpeedVolume: (a boolean or a file name)
    Output speed image
    argument: ``--outputSpeedVolume %s``
startingSeedsLabel: (an integer (int or long))
    Label value for Starting Seeds
    argument: ``--startingSeedsLabel %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
stoppingValue: (a float)
    Terminating value for vcl_cost function estimation
    argument: ``--stoppingValue %f``
inputTensorVolume: (an existing file name)
    Required: input tensor image file name
    argument: ``--inputTensorVolume %s``
inputStartingSeedsLabelMapVolume: (an existing file name)
    Required: input starting seeds LabelMap image file name
    argument: ``--inputStartingSeedsLabelMapVolume %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
```

Outputs:

```
outputSpeedVolume: (an existing file name)
    Output speed image
outputCostVolume: (an existing file name)
    Output vcl_cost image
```

75.6.11 gtractCreateGuideFiber

[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputFiber: (a boolean or a file name)
    Required: output guide fiber file name
    argument: ``--outputFiber %s``
inputFiber: (an existing file name)
    Required: input fiber tract file name
    argument: ``--inputFiber %s``
writeXMLPolyDataFile: (a boolean)
    Flag to make use of XML files when reading and writing vtkPolyData.
    argument: ``--writeXMLPolyDataFile ``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
numberOfPoints: (an integer (int or long))
    Number of points in output guide fiber
    argument: ``--numberOfPoints %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
```

Outputs:

```
outputFiber: (an existing file name)
    Required: output guide fiber file name
```

75.6.12 gtractFastMarchingTracking

[Link to code](#)

Wraps the executable 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:

```
[Optional]
numberOfIterations: (an integer (int or long))
    Number of iterations used for the optimization
    argument: ``--numberOfIterations %d``
trackingThreshold: (a float)
    Anisotropy threshold used for fiber tracking
    argument: ``--trackingThreshold %f``
seedThreshold: (a float)
    Anisotropy threshold used for seed selection
    argument: ``--seedThreshold %f``
maximumStepSize: (a float)
    Maximum step size to move when tracking
    argument: ``--maximumStepSize %f``
costStepSize: (a float)
    Cost image sub-voxel sampling
    argument: ``--costStepSize %f``
inputAnisotropyVolume: (an existing file name)
    Required: input anisotropy image file name
    argument: ``--inputAnisotropyVolume %s``
minimumStepSize: (a float)
    Minimum step size to move when tracking
    argument: ``--minimumStepSize %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
outputTract: (a boolean or a file name)
    Required: name of output vtkPolydata file containing tract lines and
    the point data collected along them.
    argument: ``--outputTract %s``
inputCostVolume: (an existing file name)
    Required: input vcl_cost image file name
    argument: ``--inputCostVolume %s``
startingSeedsLabel: (an integer (int or long))
    Label value for Starting Seeds
    argument: ``--startingSeedsLabel %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
writeXMLPolyDataFile: (a boolean)
    Flag to make use of the XML format for vtkPolyData fiber tracts.
    argument: ``--writeXMLPolyDataFile ``
inputTensorVolume: (an existing file name)
```

(continues on next page)

(continued from previous page)

```

    Required: input tensor image file name
    argument: ``--inputTensorVolume %s``
inputStartingSeedsLabelMapVolume: (an existing file name)
    Required: input starting seeds LabelMap image file name
    argument: ``--inputStartingSeedsLabelMapVolume %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``

```

Outputs:

```

outputTract: (an existing file name)
    Required: name of output vtkPolydata file containing tract lines and
    the point data collected along them.

```

75.6.13 gtractFiberTracking[Link to code](#)

Wraps the executable 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 the 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 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 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:

```
[Optional]
minimumLength: (a float)
    Minimum fiber length. Helpful for filtering invalid tracts.
    argument: ``--minimumLength %f``
trackingThreshold: (a float)
    Anisotropy threshold for fiber tracking (anisotropy values of the
    next point along the path)
    argument: ``--trackingThreshold %f``
seedThreshold: (a float)
    Anisotropy threshold for seed selection (recommended for Free fiber
    tracking)
    argument: ``--seedThreshold %f``
inputTract: (an existing file name)
    Required (for Guided fiber tracking method): guide fiber in
    vtkPolydata file containing one tract line.
    argument: ``--inputTract %s``
branchingThreshold: (a float)
    Anisotropy Branching threshold (recommended for GraphSearch fiber
    tracking method)
    argument: ``--branchingThreshold %f``
trackingMethod: ('Guided' or 'Free' or 'Streamline' or 'GraphSearch')
    Fiber tracking Filter Type: Guided|Free|Streamline|GraphSearch
    argument: ``--trackingMethod %s``
maximumBranchPoints: (an integer (int or long))
    Maximum branch points (recommended for GraphSearch fiber tracking
    method)
    argument: ``--maximumBranchPoints %d``
randomSeed: (an integer (int or long))
    Random number generator seed
    argument: ``--randomSeed %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
curvatureThreshold: (a float)
    Curvature threshold in degrees (recommended for Free fiber tracking)
    argument: ``--curvatureThreshold %f``
```

(continues on next page)

(continued from previous page)

```

maximumGuideDistance: (a float)
    Maximum distance for using the guide fiber direction
    argument: ``--maximumGuideDistance %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)
    argument: ``--startingSeedsLabel %d``
useLoopDetection: (a boolean)
    Flag to make use of loop detection.
    argument: ``--useLoopDetection``
useTend: (a boolean)
    Flag to make use of Tend F and Tend G parameters.
    argument: ``--useTend``
maximumLength: (a float)
    Maximum fiber length (voxels)
    argument: ``--maximumLength %f``
inputTensorVolume: (an existing file name)
    Required (for Free, Streamline, GraphSearch, and Guided fiber
    tracking methods): input tensor image file name
    argument: ``--inputTensorVolume %s``
tendF: (a float)
    Tend F parameter
    argument: ``--tendF %f``
inputStartingSeedsLabelMapVolume: (an existing file name)
    Required (for Free, Streamline, GraphSearch, and Guided fiber
    tracking methods): input starting seeds LabelMap image file name
    argument: ``--inputStartingSeedsLabelMapVolume %s``
inputAnisotropyVolume: (an existing file name)
    Required (for Free, Streamline, GraphSearch, and Guided fiber
    tracking methods): input anisotropy image file name
    argument: ``--inputAnisotropyVolume %s``
inputEndingSeedsLabelMapVolume: (an existing file name)
    Required (for Streamline, GraphSearch, and Guided fiber tracking
    methods): input ending seeds LabelMap image file name
    argument: ``--inputEndingSeedsLabelMapVolume %s``
tendG: (a float)
    Tend G parameter
    argument: ``--tendG %f``
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.
    argument: ``--outputTract %s``
useRandomWalk: (a boolean)
    Flag to use random walk.
    argument: ``--useRandomWalk``
stepSize: (a float)
    Fiber tracking step size
    argument: ``--stepSize %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)
    argument: ``--endingSeedsLabel %d``
writeXMLPolyDataFile: (a boolean)
    Flag to make use of the XML format for vtkPolyData fiber tracts.
    argument: ``--writeXMLPolyDataFile``
numberOfThreads: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
guidedCurvatureThreshold: (a float)
    Guided Curvature Threshold (Degrees)
    argument: ``--guidedCurvatureThreshold %f``
branchingAngle: (a float)
    Branching angle in degrees (recommended for GraphSearch fiber
    tracking method)
    argument: ``--branchingAngle %f``

```

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.

```

75.6.14 gtractImageConformity[Link to code](#)

Wraps the executable 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:

```

[Optional]
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Required: input file containing the signed short image to reorient
    without resampling.
    argument: ``--inputVolume %s``
inputReferenceVolume: (an existing file name)
    Required: input file containing the standard image to clone the
    characteristics of.
    argument: ``--inputReferenceVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: name of output Nrrd or Nifti file containing the

```

(continues on next page)

(continued from previous page)

```
reoriented image in reference image space.
argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
Required: name of output Nrrd or Nifti file containing the
reoriented image in reference image space.
```

75.6.15 gtractInvertBSplineTransform[Link to code](#)

Wraps the executable 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:

```
[Optional]
landmarkDensity: (a list of items which are an integer (int or long))
    Number of landmark subdivisions in all 3 directions
    argument: ``--landmarkDensity %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputReferenceVolume: (an existing file name)
    Required: input image file name to exemplify the anatomical space to
    interpolate over.
    argument: ``--inputReferenceVolume %s``
inputTransform: (an existing file name)
    Required: input B-Spline transform file name
    argument: ``--inputTransform %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputTransform: (a boolean or a file name)
    Required: output transform file name
    argument: ``--outputTransform %s``
```

Outputs:

```
outputTransform: (an existing file name)
Required: output transform file name
```


75.6.16 gtractInvertDisplacementField

[Link to code](#)

Wraps the executable 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:

```
[Optional]
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
deformationImage: (an existing file name)
    Required: Displacement field image
    argument: ``--deformationImage %s``
outputVolume: (a boolean or a file name)
    Required: Output deformation field
    argument: ``--outputVolume %s``
subsamplingFactor: (an integer (int or long))
    Subsampling factor for the deformation field
    argument: ``--subsamplingFactor %d``
baseImage: (an existing file name)
    Required: base image used to define the size of the inverse field
    argument: ``--baseImage %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
outputVolume: (an existing file name)
    Required: Output deformation field
```

75.6.17 gtractInvertRigidTransform

[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputTransform: (a boolean or a file name)
    Required: output transform file name
    argument: ``--outputTransform %s``
inputTransform: (an existing file name)
    Required: input rigid transform file name
    argument: ``--inputTransform %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
outputTransform: (an existing file name)
    Required: output transform file name
```

75.6.18 gtractResampleAnisotropy

[Link to code](#)

Wraps the executable 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:

```
[Optional]
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputTransform: (an existing file name)
    Required: input Rigid OR Bspline transform file name
    argument: ``--inputTransform %s``
inputAnisotropyVolume: (an existing file name)
    Required: input file containing the anisotropy image
    argument: ``--inputAnisotropyVolume %s``
transformType: ('Rigid' or 'B-Spline')
```

(continues on next page)

(continued from previous page)

```

    Transform type: Rigid, B-Spline
    argument: ``--transformType %s``
inputAnatomicalVolume: (an existing file name)
    Required: input file containing the anatomical image whose
    characteristics will be cloned.
    argument: ``--inputAnatomicalVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the resampled
    transformed anisotropy image.
    argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Required: name of output NRRD file containing the resampled
    transformed anisotropy image.

```

75.6.19 gtractResampleB0[Link to code](#)

Wraps the executable 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:

```

[Optional]
vectorIndex: (an integer (int or long))
    Index in the diffusion weighted image set for the B0 image
    argument: ``--vectorIndex %d``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Required: input file containing the 4D image
    argument: ``--inputVolume %s``
inputAnatomicalVolume: (an existing file name)
    Required: input file containing the anatomical image defining the
    origin, spacing and size of the resampled image (template)
    argument: ``--inputAnatomicalVolume %s``

```

(continues on next page)

(continued from previous page)

```

transformType: ('Rigid' or 'B-Spline')
    Transform type: Rigid, B-Spline
    argument: ``--transformType %s``
inputTransform: (an existing file name)
    Required: input Rigid OR Bspline transform file name
    argument: ``--inputTransform %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the resampled input
    image.
    argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Required: name of output NRRD file containing the resampled input
    image.

```

75.6.20 gtractResampleCodeImage**Link to code**

Wraps the executable 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:

```

[Optional]
inputReferenceVolume: (an existing file name)
    Required: input file containing the standard image to clone the
    characteristics of.
    argument: ``--inputReferenceVolume %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
transformType: ('Rigid' or 'Affine' or 'B-Spline' or 'Inverse-B-
    Spline' or 'None')
    Transform type: Rigid or Inverse-B-Spline
    argument: ``--transformType %s``
inputTransform: (an existing file name)
    Required: input Rigid or Inverse-B-Spline transform file name

```

(continues on next page)

(continued from previous page)

```

        argument: ``--inputTransform %s``
inputCodeVolume: (an existing file name)
    Required: input file containing the code image
        argument: ``--inputCodeVolume %s``
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the resampled code
        image in acquisition space.
        argument: ``--outputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

```

outputVolume: (an existing file name)
    Required: name of output NRRD file containing the resampled code
        image in acquisition space.

```

75.6.21 gtractResampleDWIInPlace[Link to code](#)

Wraps the executable 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:

```

[Optional]
debugLevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
        argument: ``--debugLevel %d``
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.
        argument: ``--imageOutputSize %s``
warpDWITransform: (an existing file name)
    Optional: transform file to warp gradient volumes.
        argument: ``--warpDWITransform %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
        argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
        argument: ``%s``
inputVolume: (an existing file name)
    Required: input image is a 4D NRRD image.
        argument: ``--inputVolume %s``
outputResampledB0: (a boolean or a file name)

```

(continues on next page)

(continued from previous page)

```

Convenience function for extracting the first index location
(assumed to be the B0)
argument: ``--outputResampledB0 %s``
referenceVolume: (an existing file name)
    If provided, resample to the final space of the referenceVolume 3D
    data set.
argument: ``--referenceVolume %s``
inputTransform: (an existing file name)
    Required: transform file derived from rigid registration of b0 image
    to reference structural image.
argument: ``--inputTransform %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
Environment variables
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.
argument: ``--outputVolume %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.

```

75.6.22 gtractResampleFibers[Link to code](#)

Wraps the executable 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:

```

[Optional]
writeXMLPolyDataFile: (a boolean)
    Flag to make use of the XML format for vtkPolyData fiber tracts.
    argument: ``--writeXMLPolyDataFile ``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

    Additional parameters to the command
    argument: ``%s``
inputReverseDeformationFieldVolume: (an existing file name)
    Required: input reverse deformation field image file name
    argument: ``--inputReverseDeformationFieldVolume %s``
inputTract: (an existing file name)
    Required: name of input vtkPolydata file containing tract lines.
    argument: ``--inputTract %s``
inputForwardDeformationFieldVolume: (an existing file name)
    Required: input forward deformation field image file name
    argument: ``--inputForwardDeformationFieldVolume %s``
environ: (a dictionary with keys which are a 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
outputTract: (a boolean or a file name)
    Required: name of output vtkPolydata file containing tract lines and
    the point data collected along them.
    argument: ``--outputTract %s``

```

Outputs:

```

outputTract: (an existing file name)
    Required: name of output vtkPolydata file containing tract lines and
    the point data collected along them.

```

75.6.23 gtractTensor[Link to code](#)

Wraps the executable 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:

```

[Optional]
maskVolume: (an existing file name)
    Mask Image, if maskProcessingMode is ROI
    argument: ``--maskVolume %s``
medianFilterSize: (a list of items which are an integer (int or
    long))
    Median filter radius in all 3 directions
    argument: ``--medianFilterSize %s``

```

(continues on next page)

(continued from previous page)

```

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.
    argument: ``--inputVolume %s``
applyMeasurementFrame: (a boolean)
    Flag to apply the measurement frame to the gradient directions
    argument: ``--applyMeasurementFrame ``
ignoreIndex: (a list of items which are an integer (int or long))
    Ignore diffusion gradient index. Used to remove specific gradient
    directions with artifacts.
    argument: ``--ignoreIndex %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
    argument: ``--maskProcessingMode %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
size: (a float)
    Isotropic voxel size to resample to
    argument: ``--size %f``
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.
    argument: ``--backgroundSuppressingThreshold %d``
resampleIsotropic: (a boolean)
    Flag to resample to isotropic voxels. Enabling this feature is
    recommended if fiber tracking will be performed.
    argument: ``--resampleIsotropic ``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
b0Index: (an integer (int or long))
    Index in input vector index to extract
    argument: ``--b0Index %d``
outputVolume: (a boolean or a file name)
    Required: name of output NRRD file containing the Tensor vector
    image
    argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Required: name of output NRRD file containing the Tensor vector

```

(continues on next page)

(continued from previous page)

image

75.6.24 gtractTransformToDisplacementField

[Link to code](#)

Wraps the executable 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 Ingahalikar, and Greg Harris

acknowledgements: Funding for this version of the GTRACT program was provided by NIH/NINDS R01NS050568-01A2S1

Inputs:

```
[Optional]
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outputDeformationFieldVolume: (a boolean or a file name)
    Output deformation field
    argument: ``--outputDeformationFieldVolume %s``
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.
    argument: ``--inputReferenceVolume %s``
inputTransform: (an existing file name)
    Input Transform File Name
    argument: ``--inputTransform %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
outputDeformationFieldVolume: (an existing file name)
    Output deformation field
```

75.7 interfaces.semtools.diffusion.maxcurvature

75.7.1 maxcurvature

[Link to code](#)

Wraps the executable command “ maxcurvature “.

title: MaxCurvature-Hessian (DTIPProcess)

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/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 dependancies on boost and a fortran compiler.

Inputs:

```
[Optional]
image: (an existing file name)
      FA Image
      argument: ``--image %s``
output: (a boolean or a file name)
      Output File
      argument: ``--output %s``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
verbose: (a boolean)
      produce verbose output
      argument: ``--verbose ``
sigma: (a float)
      Scale of Gradients
      argument: ``--sigma %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', nipype default value: {})
          Environment variables
```

Outputs:

```
output: (an existing file name)
      Output File
```

75.8 interfaces.semtools.diffusion.tractography.commandlineonly

75.8.1 fiberstats

[Link to code](#)

Wraps the executable command “ fiberstats “.

title: FiberStats (DTIPProcess)

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:

```
[Optional]
fiber_file: (an existing file name)
    DTI Fiber File
    argument: ``--fiber_file %s``
verbose: (a boolean)
    produce verbose output
    argument: ``--verbose``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

None

75.9 interfaces.semtools.diffusion.tractography.fiberprocess

75.9.1 fiberprocess

[Link to code](#)

Wraps the executable 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:

```
[Optional]
voxel_label: (an integer (int or long))
    Label for voxelized fiber
    argument: ``--voxel_label %d``
tensor_volume: (an existing file name)
    Interpolate tensor values from the given field
    argument: ``--tensor_volume %s``
fiber_radius: (a float)
    set radius of all fibers to this value
    argument: ``--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.
    argument: ``--h_field %s``
voxelize_count_fibers: (a boolean)
    Count number of fibers per-voxel instead of just setting to 1
    argument: ``--voxelize_count_fibers``
fiber_output: (a boolean or a file name)
    Output fiber file. May be warped or updated with new data depending
    on other options used.
    argument: ``--fiber_output %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
displacement_field: (an existing file name)
    Displacement Field for warp and statistics lookup. If this option is
    used tensor-volume must also be specified.
    argument: ``--displacement_field %s``
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
    argument: ``--voxelize %s``
saveProperties: (a boolean)
    save the tensor property as scalar data into the vtk (only works for
    vtk fiber files).
    argument: ``--saveProperties``
index_space: (a boolean)
    Use index-space for fiber output coordinates, otherwise us world
    space for fiber output coordinates (from tensor file).
    argument: ``--index_space``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
no_warp: (a boolean)
    Do not warp the geometry of the tensors only obtain the new
    statistics.
    argument: ``--no_warp``
noDataChange: (a boolean)
```

(continues on next page)

(continued from previous page)

```

    Do not change data ???
    argument: ``--noDataChange ``
verbose: (a boolean)
    produce verbose output
    argument: ``--verbose ``
fiber_file: (an existing file name)
    DTI fiber file
    argument: ``--fiber_file %s``

```

Outputs:

```

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
fiber_output: (an existing file name)
    Output fiber file. May be warped or updated with new data depending
    on other options used.

```

75.10 interfaces.semtools.diffusion.tractography.fibertrack

75.10.1 fibertrack

Link to code

Wraps the executable command “ fibertrack “.

title: FiberTrack (DTIPProcess)

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 dependancies on boost and a fortran compiler.

Inputs:

```

[Optional]
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.
    argument: ``--output_fiber_file %s``
max_angle: (a float)
    Maximum angle of change in radians
    argument: ``--max_angle %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
forbidden_label: (an integer (int or long))
    Forbidden label
    argument: ``--forbidden_label %d``
really_verbose: (a boolean)
    Follow detail of fiber tracking algorithm
    argument: ``--really_verbose``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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.
    argument: ``--target_label %d``
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
    argument: ``--whole_brain``
verbose: (a boolean)
    produce verbose output
    argument: ``--verbose``
step_size: (a float)
    Step size in mm for the tracking algorithm
    argument: ``--step_size %f``
min_fa: (a float)
    The minimum FA threshold to continue tractography
    argument: ``--min_fa %f``
force: (a boolean)
    Ignore sanity checks.
    argument: ``--force``
input_tensor_file: (an existing file name)
    Tensor Image
    argument: ``--input_tensor_file %s``
input_roi_file: (an existing file name)
    The filename of the image which contains the labels used for seeding
    and constraining the algorithm.
    argument: ``--input_roi_file %s``
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
    argument: ``--source_label %d``

```

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.
```

75.11 interfaces.semtools.diffusion.tractography.ukftractography

75.11.1 UKFTractography

[Link to code](#)

Wraps the executable 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 Rathi, 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 Rathi), R01 MH092862 (PIs Westin and Verma), U01 NS083223 (PI Westin), R01 MH074794 (PI Westin) and P41 EB015902 (PI Kikinis).

Inputs:

```
[Optional]
recordFA: (a boolean)
    Whether to store FA. Attaches field 'FA', and 'FA2' for 2-tensor
    case to fiber.
    argument: ``--recordFA ``
Q1: (a float)
    Process noise for eigenvalues
    argument: ``--Q1 %f``
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)..
    argument: ``--seedsPerVoxel %d``
writeAsciiTracts: (a boolean)
    Write tract file as a VTK binary data file
    argument: ``--writeAsciiTracts ``
maskFile: (an existing file name)
    Mask for diffusion tractography
    argument: ``--maskFile %s``
recordNMSE: (a boolean)
    Whether to store NMSE. Attaches field 'NMSE' to fiber.
    argument: ``--recordNMSE ``
writeUncompressedTracts: (a boolean)
    Write tract file as a VTK uncompressed data file
    argument: ``--writeUncompressedTracts ``
tractsWithSecondTensor: (a boolean or a file name)
    Tracts generated, with second tensor output (if there is one)
    argument: ``--tractsWithSecondTensor %s``
minBranchingAngle: (a float)
    Minimum branching angle, in degrees. When using multiple tensors, a
```

(continues on next page)

(continued from previous page)

```

        new branch will be created when the tensors' major directions form
        an angle between (minBranchingAngle, maxBranchingAngle)
        argument: ``--minBranchingAngle %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
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
        argument: ``--seedsFile %s``
recordCovariance: (a boolean)
        Whether to store the covariance. Will generate field 'covariance' in
        fiber.
        argument: ``--recordCovariance``
seedFALimit: (a float)
        Seed points whose FA are below this value are excluded
        argument: ``--seedFALimit %f``
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.
        argument: ``--recordTensors``
Rs: (a float)
        Measurement noise
        argument: ``--Rs %f``
stepLength: (a float)
        Step length of tractography, in millimeters
        argument: ``--stepLength %f``
minFA: (a float)
        Abort the tractography when the Fractional Anisotropy is less than
        this value
        argument: ``--minFA %f``
freeWater: (a boolean)
        Adds a term for free water difusion to the model. (Note for experts:
        if checked, the 1T simple model is forced)
        argument: ``--freeWater``
labels: (a list of items which are an integer (int or long))
        A vector of the ROI labels to be used
        argument: ``--labels %s``
recordLength: (a float)
        Record length of tractography, in millimeters
        argument: ``--recordLength %f``
minGA: (a float)
        Abort the tractography when the Generalized Anisotropy is less than
        this value
        argument: ``--minGA %f``
recordFreeWater: (a boolean)
        Whether to store the fraction of free water. Attaches field
        'FreeWater' to fiber.
        argument: ``--recordFreeWater``
recordTrace: (a boolean)
        Whether to store Trace. Attaches field 'Trace', and 'Trace2' for
        2-tensor case to fiber.
        argument: ``--recordTrace``
maxHalfFiberLength: (a float)

```

(continues on next page)

(continued from previous page)

```

    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
    argument: ``--maxHalfFiberLength %f``
dwiFile: (an existing file name)
    Input DWI volume
    argument: ``--dwiFile %s``
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
    argument: ``--storeGlyphs``
recordState: (a boolean)
    Whether to attach the states to the fiber. Will generate field
    'state'.
    argument: ``--recordState``
Qm: (a float)
    Process noise for angles/direction
    argument: ``--Qm %f``
tracts: (a boolean or a file name)
    Tracts generated, with first tensor output
    argument: ``--tracts %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.
    argument: ``--numThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
numTensor: ('1' or '2')
    Number of tensors used
    argument: ``--numTensor %s``
fullTensorModel: (a boolean)
    Whether to use the full tensor model. If unchecked, use the default
    simple tensor model
    argument: ``--fullTensorModel``
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
    argument: ``--maxBranchingAngle %f``
Qw: (a float)
    Process noise for free water weights, ignored if no free water
    estimation
    argument: ``--Qw %f``

```

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)

```

75.12 interfaces.semtools.featurecreator

75.12.1 GenerateCsfClippedFromClassifiedImage

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputCassifiedVolume: (an existing file name)
    Required: input tissue label image
    argument: ``--inputCassifiedVolume %s``
outputVolume: (a boolean or a file name)
    Required: output image
    argument: ``--outputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

75.13 interfaces.semtools.filtering.denoising

75.13.1 UnbiasedNonLocalMeans

[Link to code](#)

Wraps the executable 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:

```
[Optional]
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)
    argument: ``--ps %f``
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
    argument: ``--hp %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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).
    argument: ``--rs %s``
inputVolume: (an existing file name)
    Input MRI volume.
    argument: ``%s``, position: -2
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
    argument: ``--rc %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.
    argument: ``--sigma %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
outputVolume: (a boolean or a file name)
    Output (filtered) MRI volume.
    argument: ``%s``, position: -1
```

Outputs:

```
outputVolume: (an existing file name)
    Output (filtered) MRI volume.
```

75.14 interfaces.semtools.filtering.featuredetection

75.14.1 CannyEdge

[Link to code](#)

Wraps the executable 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:

```
[Optional]
variance: (a float)
    Variance and Maximum error are used in the Gaussian smoothing of the
    input image. See itkDiscreteGaussianImageFilter for information on
    these parameters.
    argument: ``--variance %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--lowerThreshold %f``
inputVolume: (an existing file name)
    Required: input tissue label image
    argument: ``--inputVolume %s``
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.
    argument: ``--upperThreshold %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
outputVolume: (a boolean or a file name)
    Required: output image
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

75.14.2 CannySegmentationLevelSetImageFilter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
cannyThreshold: (a float)
    Canny Threshold Value
    argument: ``--cannyThreshold %f``
cannyVariance: (a float)
    Canny variance
    argument: ``--cannyVariance %f``
outputSpeedVolume: (a boolean or a file name)
    argument: ``--outputSpeedVolume %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
maxIterations: (an integer (int or long))
    The
    argument: ``--maxIterations %d``
inputVolume: (an existing file name)
    argument: ``--inputVolume %s``
advectionWeight: (a float)
    Controls the smoothness of the resulting mask, small number are more
    smooth, large numbers allow more sharp corners.
    argument: ``--advectionWeight %f``
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).
    argument: ``--initialModelIsovalue %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
outputVolume: (a boolean or a file name)
    argument: ``--outputVolume %s``
initialModel: (an existing file name)
    argument: ``--initialModel %s``
```

Outputs:

```
outputSpeedVolume: (an existing file name)
outputVolume: (an existing file name)
```

75.14.3 DilateImage

[Link to code](#)

Wraps the executable 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:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Required: input image
    argument: ``--inputVolume %s``
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    argument: ``--inputMaskVolume %s``
inputRadius: (an integer (int or long))
    Required: input neighborhood radius
    argument: ``--inputRadius %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
outputVolume: (a boolean or a file name)
    Required: output image
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

75.14.4 DilateMask

[Link to code](#)

Wraps the executable 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:

```
[Optional]
lowerThreshold: (a float)
    Required: lowerThreshold value
    argument: ``--lowerThreshold %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputBinaryVolume: (an existing file name)
    Required: input brain mask image
    argument: ``--inputBinaryVolume %s``
inputVolume: (an existing file name)
    Required: input image
    argument: ``--inputVolume %s``
```

(continues on next page)

(continued from previous page)

```

outputVolume: (a boolean or a file name)
    Required: output image
    argument: ``--outputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
sizeStructuralElement: (an integer (int or long))
    size of structural element. sizeStructuralElement=1 means that 3x3x3
    structuring element for 3D
    argument: ``--sizeStructuralElement %d``

```

Outputs:

```

outputVolume: (an existing file name)
    Required: output image

```

75.14.5 DistanceMaps[Link to code](#)

Wraps the executable command “DistanceMaps”.

title: Maurer 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:

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputLabelVolume: (an existing file name)
    Required: input tissue label image
    argument: ``--inputLabelVolume %s``
inputTissueLabel: (an integer (int or long))
    Required: input integer value of tissue type used to calculate
    distance
    argument: ``--inputTissueLabel %d``
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    argument: ``--inputMaskVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: output image
    argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Required: output image

```

75.14.6 DumpBinaryTrainingVectors

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputHeaderFilename: (an existing file name)
    Required: input header file name
    argument: ``--inputHeaderFilename %s``
inputVectorFilename: (an existing file name)
    Required: input vector filename
    argument: ``--inputVectorFilename %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

None

75.14.7 ErodeImage

[Link to code](#)

Wraps the executable 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:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Required: input image
    argument: ``--inputVolume %s``
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    argument: ``--inputMaskVolume %s``
inputRadius: (an integer (int or long))
    Required: input neighborhood radius
```

(continues on next page)

(continued from previous page)

```

argument: ``--inputRadius %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
outputVolume: (a boolean or a file name)
Required: output image
argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
Required: output image

```

75.14.8 FlippedDifference[Link to code](#)

Wraps the executable 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:

```

[Optional]
inputVolume: (an existing file name)
Required: input image
argument: ``--inputVolume %s``
inputMaskVolume: (an existing file name)
Required: input brain mask image
argument: ``--inputMaskVolume %s``
outputVolume: (a boolean or a file name)
Required: output image
argument: ``--outputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
          of class 'str' and with values which are a bytes or None or a
          value of class 'str', nipy default value: {})
Environment variables
args: (a unicode string)
Additional parameters to the command
argument: ``%s``

```

Outputs:

```

outputVolume: (an existing file name)
Required: output image

```

75.14.9 GenerateBrainClippedImage[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputFileName: (a boolean or a file name)
    (required) output file name
    argument: ``--outputFileName %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputMsk: (an existing file name)
    input volume 2, usually t2 image
    argument: ``--inputMsk %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
inputImg: (an existing file name)
    input volume 1, usually t1 image
    argument: ``--inputImg %s``
```

Outputs:

```
outputFileName: (an existing file name)
    (required) output file name
```

75.14.10 GenerateSummedGradientImage

[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputFileName: (a boolean or a file name)
    (required) output file name
    argument: ``--outputFileName %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
MaximumGradient: (a boolean)
    If set this flag, it will compute maximum gradient between two input
    volumes instead of sum of it.
```

(continues on next page)

(continued from previous page)

```

        argument: ``--MaximumGradient ``
inputVolume2: (an existing file name)
        input volume 2, usually t2 image
        argument: ``--inputVolume2 %s``
inputVolume1: (an existing file name)
        input volume 1, usually t1 image
        argument: ``--inputVolume1 %s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
Environment variables

```

Outputs:

```

outputFileName: (an existing file name)
                (required) output file name

```

75.14.11 GenerateTestImage[Link to code](#)

Wraps the executable 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:

```

[Optional]
outputVolume: (a boolean or a file name)
                (required) output file name
                argument: ``--outputVolume %s``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
upperBoundOfOutputVolume: (a float)
        argument: ``--upperBoundOfOutputVolume %f``
inputVolume: (an existing file name)
        input volume 1, usually t1 image
        argument: ``--inputVolume %s``
lowerBoundOfOutputVolume: (a float)
        argument: ``--lowerBoundOfOutputVolume %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
outputVolumeSize: (a float)
        output Volume Size
        argument: ``--outputVolumeSize %f``

```

Outputs:

```

outputVolume: (an existing file name)
                (required) output file name

```

75.14.12 GradientAnisotropicDiffusionImageFilter

[Link to code](#)

Wraps the executable command “ GradientAnisotropicDiffusionImageFilter “.

title: GradientAnisotropicDiffusionFilter

category: Filtering.FeatureDetection

description: Image Smoothing using Gradient Anisotropic Diffusion Filter

contributor: This tool was developed by Eun Young Kim by modifying ITK Example

Inputs:

```
[Optional]
numberOfIterations: (an integer (int or long))
    Optional value for number of Iterations
    argument: ``--numberOfIterations %d``
timeStep: (a float)
    Time step for diffusion process
    argument: ``--timeStep %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
conductance: (a float)
    Conductance for diffusion process
    argument: ``--conductance %f``
inputVolume: (an existing file name)
    Required: input image
    argument: ``--inputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Required: output image
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

75.14.13 HammerAttributeCreator

[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputVolumeBase: (a unicode string)
    Required: output image base name to be appended for each feature
    vector.
    argument: ``--outputVolumeBase %s``
```

(continues on next page)

(continued from previous page)

```

Strength: (a float)
    Determine Strength of Edges
    argument: ``--Strength %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputWMVolume: (an existing file name)
    Required: input white matter posterior image
    argument: ``--inputWMVolume %s``
inputCSFVolume: (an existing file name)
    Required: input CSF posterior image
    argument: ``--inputCSFVolume %s``
Scale: (an integer (int or long))
    Determine Scale of Ball
    argument: ``--Scale %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
inputGMVolume: (an existing file name)
    Required: input grey matter posterior image
    argument: ``--inputGMVolume %s``

```

Outputs:

None

75.14.14 NeighborhoodMean

[Link to code](#)

Wraps the executable 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:

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Required: input image
    argument: ``--inputVolume %s``
inputMaskVolume: (an existing file name)
    Required: input brain mask image
    argument: ``--inputMaskVolume %s``
inputRadius: (an integer (int or long))
    Required: input neighborhood radius
    argument: ``--inputRadius %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: {})

```

(continues on next page)

(continued from previous page)

```

Environment variables
outputVolume: (a boolean or a file name)
Required: output image
argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
Required: output image

```

75.14.15 NeighborhoodMedian[Link to code](#)

Wraps the executable 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:

```

[Optional]
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
inputVolume: (an existing file name)
      Required: input image
      argument: ``--inputVolume %s``
inputMaskVolume: (an existing file name)
      Required: input brain mask image
      argument: ``--inputMaskVolume %s``
inputRadius: (an integer (int or long))
      Required: input neighborhood radius
      argument: ``--inputRadius %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
outputVolume: (a boolean or a file name)
      Required: output image
      argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
Required: output image

```

75.14.16 STAPLEAnalysis[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputVolume: (a boolean or a file name)
    Required: output image
    argument: ``--outputVolume %s``
inputDimension: (an integer (int or long))
    Required: input image Dimension 2 or 3
    argument: ``--inputDimension %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', nipyte default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputLabelVolume: (a list of items which are an existing file name)
    Required: input label volume
    argument: ``--inputLabelVolume %s...``
```

Outputs:

```
outputVolume: (an existing file name)
    Required: output image
```

75.14.17 TextureFromNoiseImageFilter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputVolume: (an existing file name)
    Required: input image
    argument: ``--inputVolume %s``
outputVolume: (a boolean or a file name)
    Required: output image
    argument: ``--outputVolume %s``
inputRadius: (an integer (int or long))
    Required: input neighborhood radius
    argument: ``--inputRadius %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', nipyte default value: {})
    Environment variables
args: (a unicode string)
```

(continues on next page)

(continued from previous page)

```
Additional parameters to the command
argument: ``%s``
```

Outputs:

```
outputVolume: (an existing file name)
Required: output image
```

75.14.18 TextureMeasureFilter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
insideROIValue: (a float)
    argument: ``--insideROIValue %f``
distance: (an integer (int or long))
    argument: ``--distance %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outputFilename: (a boolean or a file name)
    argument: ``--outputFilename %s``
inputVolume: (an existing file name)
    argument: ``--inputVolume %s``
inputMaskVolume: (an existing file name)
    argument: ``--inputMaskVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
outputFilename: (an existing file name)
```

75.15 interfaces.semtools.legacy.registration

75.15.1 scalartransform

[Link to code](#)

Wraps the executable command “ scalartransform “.

title: ScalarTransform (DTIPProcess)

category: Legacy.Registration

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:

```
[Optional]
interpolation: ('nearestneighbor' or 'linear' or 'cubic')
    Interpolation type (nearestneighbor, linear, cubic)
    argument: ``--interpolation %s``
input_image: (an existing file name)
    Image to transform
    argument: ``--input_image %s``
transformation: (a boolean or a file name)
    Output file for transformation parameters
    argument: ``--transformation %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
h_field: (a boolean)
    The deformation is an h-field.
    argument: ``--h_field ``
invert: (a boolean)
    Invert tranform before applying.
    argument: ``--invert ``
output_image: (a boolean or a file name)
    The transformed image
    argument: ``--output_image %s``
deformation: (an existing file name)
    Deformation field.
    argument: ``--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
```

Outputs:

```
output_image: (an existing file name)
    The transformed image
transformation: (an existing file name)
    Output file for transformation parameters
```

75.16 interfaces.semtools.registration.brainsfit

75.16.1 BRAINSFit

[Link to code](#)

Wraps the executable 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 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:

```
[Optional]
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.
    argument: ``--failureExitCode %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.
    argument: ``--numberOfIterations %s``
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
    argument: ``--reproportionScale %f``
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.
    argument: ``--histogramMatch ``
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.
    argument: ``--outputMovingVolumeROI %s``
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.
    argument: ``--scaleOutputValues ``
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.
    argument: ``--useRigid ``
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
    argument: ``--skewScale %f``
```

(continues on next page)

(continued from previous page)

```

debugLevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
    argument: ``--debugLevel %d``

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.
    argument: ``--splineGridSize %s``

metricSamplingStrategy: ('Random')
    It defines the method that registration filter uses to sample the
    input fixed image. Only Random is supported for now.
    argument: ``--metricSamplingStrategy %s``

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.
    argument: ``--samplingPercentage %f``

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.
    argument: ``--strippedOutputTransform %s``

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.
    argument: ``--ROIAutoClosingSize %f``

maximumNumberOfEvaluations: (an integer (int or long))
    Maximum number of evaluations for line search in lbfgsb optimizer.
    argument: ``--maximumNumberOfEvaluations %d``

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.
    argument: ``--interpolationMode %s``

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).
    argument: ``--medianFilterSize %s``

maximumStepLength: (a float)

```

(continues on next page)

(continued from previous page)

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.
argument: ``--maximumStepLength %f``

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.
argument: ``--translationScale %f``

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.
argument: ``--numberOfSamples %d``

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.
argument: ``--transformType %s``

numberOfMatchPoints: (an integer (int or long))
Number of histogram match points used for mutual information metric estimation.
argument: ``--numberOfMatchPoints %d``

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).
argument: ``--fixedVolume2 %s``

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.
argument: ``--useROIBSpline``

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.
argument: ``--initialTransform %s``

numberOfThreads: (an integer (int or long))
Explicitly specify the maximum number of threads to use. (default is auto-detected)
argument: ``--numberOfThreads %d``

args: (a unicode string)
Additional parameters to the command
argument: ``%s``

movingVolume: (an existing file name)
Input moving image (this image will be transformed into the fixed image space).
argument: ``--movingVolume %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

(continues on next page)

(continued from previous page)

```

        for low accuracy; 1e+7 for moderate accuracy and 1e+1 for extremely
        high accuracy. 1e+9 seems to work well.,
        argument: ``--costFunctionConvergenceFactor %f``
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    argument: ``--promptUser ``
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
    argument: ``--movingVolumeTimeIndex %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.
    argument: ``--gui ``
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels used for mutual information metric
    estimation.
    argument: ``--numberOfHistogramBins %d``
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.
    argument: ``--useScaleVersor3D ``
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.
    argument: ``--useBSpline ``
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.
    argument: ``--fixedBinaryVolume %s``
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.
    argument: ``--useScaleSkewVersor3D ``
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.
    argument: ``--useSyN ``
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.
    argument: ``--relaxationFactor %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

```

(continues on next page)

(continued from previous page)

```

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.
    argument: ``--outputTransform %s``
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.
    argument: ``--writeOutputTransformInFloat ``
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.,
    argument: ``--projectedGradientTolerance %f``
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.
    argument: ``--maskInferiorCutOffFromCenter %f``
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).
    argument: ``--linearTransform %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.
    argument: ``--minimumStepLength %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.
    argument: ``--fixedVolumeTimeIndex %d``
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.,
    argument: ``--maxBSplineDisplacement %f``
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
    argument: ``--logFileReport %s``
writeTransformOnFailure: (a boolean)
    Flag to save the final transform even if the numberOfIterations are

```

(continues on next page)

(continued from previous page)

```

    reached without convergence. (Intended for use when
    --failureExitCode 0 )
    argument: ``--writeTransformOnFailure ``
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).
    argument: ``--bsplineTransform %s``
fixedVolume: (an existing file name)
    Input fixed image (the moving image will be transformed into this
    image space).
    argument: ``--fixedVolume %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).
    argument: ``--outputVolume %s``
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.
    argument: ``--useComposite ``
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.
    argument: ``--maskProcessingMode %s``
maximumNumberOfCorrections: (an integer (int or long))
    Maximum number of corrections in lbfgsb optimizer.
    argument: ``--maximumNumberOfCorrections %d``
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)
    argument: ``--costMetric %s``
initializeTransformMode: ('Off' or 'useMomentsAlign' or
    'useCenterOfHeadAlign' or 'useGeometryAlign' or
    'useCenterOfROIAlign')
    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.
    argument: ``--initializeTransformMode %s``
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.
    argument: ``--removeIntensityOutliers %f``
backgroundFillValue: (a float)

```

(continues on next page)

(continued from previous page)

```

        This value will be used for filling those areas of the output image
        that have no corresponding voxels in the input moving image.
        argument: ``--backgroundFillValue %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.
        argument: ``--useAffine``
    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.
        argument: ``--outputFixedVolumeROI %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).
        argument: ``--movingVolume2 %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.
        argument: ``--initializeRegistrationByCurrentGenericTransform``
    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.
        argument: ``--movingBinaryVolume %s``
    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.
        argument: ``--ROIAutoDilateSize %f``
    outputVolumePixelType: ('float' or 'short' or 'ushort' or 'int' or
        'uint' or 'uchar')
        Data type for representing a voxel of the Output Volume.
        argument: ``--outputVolumePixelType %s``

```

Outputs:

```

    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.
    bsplineTransform: (an existing file name)
        (optional) Output estimated transform - in case the computed

```

(continues on next page)

(continued from previous page)

```

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).
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.
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.
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).
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.

```

75.17 interfaces.semtools.registration.brainsresample

75.17.1 BRAINSResample

[Link to code](#)

Wraps the executable 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:

```

[Optional]
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
argument: ``--interpolationMode %s``

```

(continues on next page)

(continued from previous page)

```

defaultValue: (a float)
    Default voxel value
    argument: ``--defaultValue %f``
inputVolume: (an existing file name)
    Image To Warp
    argument: ``--inputVolume %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.
    argument: ``--referenceVolume %s``
environ: (a dictionary with keys which are a 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
inverseTransform: (a boolean)
    True/False is to compute inverse of given transformation. Default is
    false
    argument: ``--inverseTransform ``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--pixelType %s``
deformationVolume: (an existing file name)
    Displacement Field to be used to warp the image (ITKv3 or earlier)
    argument: ``--deformationVolume %s``
warpTransform: (an existing file name)
    Filename for the BRAINSFit transform (ITKv3 or earlier) or composite
    transform file (ITKv4)
    argument: ``--warpTransform %s``
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
    argument: ``--gridSpacing %s``
outputVolume: (a boolean or a file name)
    Resulting deformed image
    argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Resulting deformed image

```

75.18 interfaces.semtools.registration.brainsresize

75.18.1 BRAINSResize

[Link to code](#)

Wraps the executable 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:

```
[Optional]
scaleFactor: (a float)
    The scale factor for the image spacing.
    argument: ``--scaleFactor %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--pixelType %s``
inputVolume: (an existing file name)
    Image To Scale
    argument: ``--inputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Resulting scaled image
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Resulting scaled image
```

75.19 interfaces.semtools.registration.specialized

75.19.1 BRAINSDemonWarp

[Link to code](#)

Wraps the executable 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:

```
[Optional]
numberOfPyramidLevels: (an integer (int or long))
    Number of image pyramid levels to use in the multi-resolution
    registration.
    argument: ``--numberOfPyramidLevels %d``
max_step_length: (a float)
    Maximum length of an update vector (0: no restriction)
    argument: ``--max_step_length %f``
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels
    argument: ``--numberOfHistogramBins %d``
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)
    argument: ``--gradient_type %s``
outputDisplacementFieldVolume: (a boolean or a file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
    argument: ``--outputDisplacementFieldVolume %s``
seedForBOBF: (a list of items which are an integer (int or long))
    coordinates in all 3 directions for Seed when performing BOBF
    argument: ``--seedForBOBF %s``
outputDisplacementFieldPrefix: (a unicode string)
    Displacement field filename prefix for writing separate x, y, and z
    component images
    argument: ``--outputDisplacementFieldPrefix %s``
fixedBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Fixed image.
    argument: ``--fixedBinaryVolume %s``
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.
    argument: ``--histogramMatch ``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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)
```

(continues on next page)

(continued from previous page)

```

        argument: ``--minimumMovingPyramid %s``
numberOfMatchPoints: (an integer (int or long))
    The number of match points for histogramMatch
    argument: ``--numberOfMatchPoints %d``
numberOfBCHApproximationTerms: (an integer (int or long))
    Number of terms in the BCH expansion
    argument: ``--numberOfBCHApproximationTerms %d``
use_vanilla_dem: (a boolean)
    Run vanilla demons algorithm
    argument: ``--use_vanilla_dem``
checkerboardPatternSubdivisions: (a list of items which are an
    integer (int or long))
    Number of Checkerboard subdivisions in all 3 directions
    argument: ``--checkerboardPatternSubdivisions %s``
outputDebug: (a boolean)
    Flag to write debugging images after each step.
    argument: ``--outputDebug``
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.
    argument: ``--outputNormalized``
neighborhoodForBOBF: (a list of items which are an integer (int or
    long))
    neighborhood in all 3 directions to be included when performing BOBF
    argument: ``--neighborhoodForBOBF %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
    argument: ``--interpolationMode %s``
lowerThresholdForBOBF: (an integer (int or long))
    Lower threshold for performing BOBF
    argument: ``--lowerThresholdForBOBF %d``
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.
    argument: ``--medianFilterSize %s``
upperThresholdForBOBF: (an integer (int or long))
    Upper threshold for performing BOBF
    argument: ``--upperThresholdForBOBF %d``
fixedVolume: (an existing file name)
    Required: input fixed (target) image
    argument: ``--fixedVolume %s``
outputVolume: (a boolean or a file name)
    Required: output resampled moving image (will have the same physical
    space as the fixedVolume).
    argument: ``--outputVolume %s``
inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    Input volumes will be typecast to this format:
    float|short|ushort|int|uchar
    argument: ``--inputPixelType %s``
maskProcessingMode: ('NOMASK' or 'ROIAUTO' or 'ROI' or 'BOBF')
    What mode to use for using the masks: NOMASK|ROIAUTO|ROI|BOBF. If

```

(continues on next page)

(continued from previous page)

```

    ROIAUTO is choosen, then the mask is implicitly defined using a otsu
    forground 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.
    argument: ``--maskProcessingMode %s``
arrayOfPyramidLevelIterations: (a list of items which are an integer
    (int or long))
    The number of iterations for each pyramid level
    argument: ``--arrayOfPyramidLevelIterations %s``
registrationFilterType: ('Demons' or 'FastSymmetricForces' or
    'Diffeomorphic')
    Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic
    argument: ``--registrationFilterType %s``
outputCheckerboardVolume: (a boolean or a file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
    argument: ``--outputCheckerboardVolume %s``
initializeWithDisplacementField: (an existing file name)
    Initial deformation field vector image file name
    argument: ``--initializeWithDisplacementField %s``
gui: (a boolean)
    Display intermediate image volumes for debugging
    argument: ``--gui ``
initializeWithTransform: (an existing file name)
    Initial Transform filename
    argument: ``--initializeWithTransform %s``
backgroundFillValue: (an integer (int or long))
    Replacement value to overwrite background when performing BOBF
    argument: ``--backgroundFillValue %d``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
movingVolume: (an existing file name)
    Required: input moving image
    argument: ``--movingVolume %s``
upFieldSmoothing: (a float)
    Smoothing sigma for the update field at each iteration
    argument: ``--upFieldSmoothing %f``
outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    outputVolume will be typecast to this format:
    float|short|ushort|int|uchar
    argument: ``--outputPixelType %s``
movingBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Moving image.
    argument: ``--movingBinaryVolume %s``
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    argument: ``--promptUser ``
minimumFixedPyramid: (a list of items which are an integer (int or
    long))
    The shrink factor for the first level of the fixed image pyramid.

```

(continues on next page)

(continued from previous page)

```

        (i.e. start at 1/16 scale, then 1/8, then 1/4, then 1/2, and finally
        full scale)
        argument: ``--minimumFixedPyramid %s``
smoothDisplacementFieldSigma: (a float)
        A gaussian smoothing value to be applied to the deformation feild at
        each iteration.
        argument: ``--smoothDisplacementFieldSigma %f``

```

Outputs:

```

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).
outputCheckerboardVolume: (an existing file name)
        Genete a checkerboard image volume between the fixedVolume and the
        deformed movingVolume.

```

75.19.2 BRAINSTransformFromFiducials[Link to code](#)

Wraps the executable 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-Docmentation-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:

```

[Optional]
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
        argument: ``--movingLandmarks %s...``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
movingLandmarksFile: (an existing file name)
        An fcsv formatted file with a list of landmark points.
        argument: ``--movingLandmarksFile %s``
fixedLandmarksFile: (an existing file name)
        An fcsv formatted file with a list of landmark points.
        argument: ``--fixedLandmarksFile %s``
transformType: ('Translation' or 'Rigid' or 'Similarity')
        Type of transform to produce
        argument: ``--transformType %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
        argument: ``--fixedLandmarks %s...``

```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
saveTransform: (a boolean or a file name)
         Save the transform that results from registration
         argument: ``--saveTransform %s``
numberOfThreads: (an integer (int or long))
         Explicitly specify the maximum number of threads to use.
         argument: ``--numberOfThreads %d``

```

Outputs:

```

saveTransform: (an existing file name)
         Save the transform that results from registration

```

75.19.3 VBRAINSDemonWarp**Link to code**

Wraps the executable 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/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:

```

[Optional]
numberOfPyramidLevels: (an integer (int or long))
         Number of image pyramid levels to use in the multi-resolution
         registration.
         argument: ``--numberOfPyramidLevels %d``
max_step_length: (a float)
         Maximum length of an update vector (0: no restriction)
         argument: ``--max_step_length %f``
numberOfHistogramBins: (an integer (int or long))
         The number of histogram levels
         argument: ``--numberOfHistogramBins %d``
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)
         argument: ``--gradient_type %s``
outputDisplacementFieldVolume: (a boolean or a file name)
         Output deformation field vector image (will have the same physical
         space as the fixedVolume).
         argument: ``--outputDisplacementFieldVolume %s``

```

(continues on next page)

(continued from previous page)

```

seedForBOBF: (a list of items which are an integer (int or long))
    coordinates in all 3 directions for Seed when performing BOBF
    argument: ``--seedForBOBF %s``
lowerThresholdForBOBF: (an integer (int or long))
    Lower threshold for performing BOBF
    argument: ``--lowerThresholdForBOBF %d``
fixedBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Fixed image.
    argument: ``--fixedBinaryVolume %s``
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.
    argument: ``--histogramMatch``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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)
    argument: ``--minimumMovingPyramid %s``
numberOfMatchPoints: (an integer (int or long))
    The number of match points for histogramMatch
    argument: ``--numberOfMatchPoints %d``
numberOfBCHApproximationTerms: (an integer (int or long))
    Number of terms in the BCH expansion
    argument: ``--numberOfBCHApproximationTerms %d``
registrationFilterType: ('Demons' or 'FastSymmetricForces' or
    'Diffeomorphic' or 'LogDemons' or 'SymmetricLogDemons')
    Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic|L
    ogDemons|SymmetricLogDemons
    argument: ``--registrationFilterType %s``
use_vanilla_dem: (a boolean)
    Run vanilla demons algorithm
    argument: ``--use_vanilla_dem``
checkerboardPatternSubdivisions: (a list of items which are an
    integer (int or long))
    Number of Checkerboard subdivisions in all 3 directions
    argument: ``--checkerboardPatternSubdivisions %s``
outputDebug: (a boolean)
    Flag to write debugging images after each step.
    argument: ``--outputDebug``
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.
    argument: ``--outputNormalized``
neighborhoodForBOBF: (a list of items which are an integer (int or
    long))
    neighborhood in all 3 directions to be included when performing BOBF
    argument: ``--neighborhoodForBOBF %s``
weightFactors: (a list of items which are a float)
    Weight fatctors for each input images
    argument: ``--weightFactors %s``

```

(continues on next page)

(continued from previous page)

```

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
    argument: ``--interpolationMode %s``
initializeWithDisplacementField: (an existing file name)
    Initial deformation field vector image file name
    argument: ``--initializeWithDisplacementField %s``
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.
    argument: ``--medianFilterSize %s``
upperThresholdForBOBF: (an integer (int or long))
    Upper threshold for performing BOBF
    argument: ``--upperThresholdForBOBF %d``
fixedVolume: (a list of items which are an existing file name)
    Required: input fixed (target) image
    argument: ``--fixedVolume %s...``
outputDisplacementFieldPrefix: (a unicode string)
    Displacement field filename prefix for writing separate x, y, and z
    component images
    argument: ``--outputDisplacementFieldPrefix %s``
inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    Input volumes will be typecast to this format:
    float|short|ushort|int|uchar
    argument: ``--inputPixelType %s``
arrayOfPyramidLevelIterations: (a list of items which are an integer
    (int or long))
    The number of iterations for each pyramid level
    argument: ``--arrayOfPyramidLevelIterations %s``
makeBOBF: (a boolean)
    Flag to make Brain-Only Background-Filled versions of the input and
    target volumes.
    argument: ``--makeBOBF ``
outputCheckerboardVolume: (a boolean or a file name)
    Generate a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
    argument: ``--outputCheckerboardVolume %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)
    argument: ``--minimumFixedPyramid %s``
gui: (a boolean)
    Display intermediate image volumes for debugging
    argument: ``--gui ``
initializeWithTransform: (an existing file name)
    Initial Transform filename
    argument: ``--initializeWithTransform %s``
backgroundFillValue: (an integer (int or long))
    Replacement value to overwrite background when performing BOBF
    argument: ``--backgroundFillValue %d``
numberOfThreads: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

        Explicitly specify the maximum number of threads to use.
        argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
movingVolume: (a list of items which are an existing file name)
    Required: input moving image
    argument: ``--movingVolume %s...``
upFieldSmoothing: (a float)
    Smoothing sigma for the update field at each iteration
    argument: ``--upFieldSmoothing %f``
outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    outputVolume will be typecast to this format:
    float|short|ushort|int|uchar
    argument: ``--outputPixelType %s``
movingBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Moving image.
    argument: ``--movingBinaryVolume %s``
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    argument: ``--promptUser ``
outputVolume: (a boolean or a file name)
    Required: output resampled moving image (will have the same physical
    space as the fixedVolume).
    argument: ``--outputVolume %s``
smoothDisplacementFieldSigma: (a float)
    A gaussian smoothing value to be applied to the deformation feild at
    each iteration.
    argument: ``--smoothDisplacementFieldSigma %f``

```

Outputs:

```

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).
outputCheckerboardVolume: (an existing file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.

```

75.20 interfaces.semtools.segmentation.specialized

75.20.1 BRAINSABC

[Link to code](#)

Wraps the executable 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:

```

[Optional]
filterMethod: ('None' or 'CurvatureFlow' or
               'GradientAnisotropicDiffusion' or 'Median')
    Filter method for preprocessing of registration
    argument: ``--filterMethod %s``
maxBiasDegree: (an integer (int or long))
    Maximum bias degree
    argument: ``--maxBiasDegree %d``
debuglevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
    argument: ``--debuglevel %d``
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.
    argument: ``--atlasToSubjectTransformType %s``
gridSize: (a list of items which are an integer (int or long))
    Grid size for atlas warping with BSplines
    argument: ``--gridSize %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.
    argument: ``--numberOfSubSamplesInEachPlugArea %s``
environ: (a dictionary with keys which are a 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
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.
    argument: ``--subjectIntermodeTransformType %s``
useKNN: (a boolean)
    Use the KNN stage of estimating posteriors.
    argument: ``--useKNN``
outputFormat: ('NIFTI' or 'Meta' or 'Nrrd')
    Output format
    argument: ``--outputFormat %s``
atlasToSubjectInitialTransform: (a boolean or a file name)
    The initial transform from atlas to the subject
    argument: ``--atlasToSubjectInitialTransform %s``
saveState: (a boolean or a file name)
    (optional) Filename to which save the final state of the
    registration
    argument: ``--saveState %s``
atlasDefinition: (an existing file name)
    Contains all parameters for Atlas
    argument: ``--atlasDefinition %s``
restoreState: (an existing file name)
    The initial state for the registration process
    argument: ``--restoreState %s``
inputVolumes: (a list of items which are an existing file name)
    The list of input image files to be segmented.
    argument: ``--inputVolumes %s...``
interpolationMode: ('BSpline' or 'NearestNeighbor' or 'WindowedSinc')

```

(continues on next page)

(continued from previous page)

```

    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.
    argument: ``--interpolationMode %s``
posteriorTemplate: (a unicode string)
    filename template for Posterior output files
    argument: ``--posteriorTemplate %s``
medianFilterSize: (a list of items which are an integer (int or
    long))
    The radius for the optional MedianImageFilter preprocessing in all 3
    directions.
    argument: ``--medianFilterSize %s``
outputLabels: (a boolean or a file name)
    Output Label Image
    argument: ``--outputLabels %s``
atlasWarpingOff: (a boolean)
    Deformable registration of atlas to subject
    argument: ``--atlasWarpingOff ``
inputVolumeTypes: (a list of items which are a unicode string)
    The list of input image types corresponding to the inputVolumes.
    argument: ``--inputVolumeTypes %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.
    argument: ``--outputVolumes %s...``
writeLess: (a boolean)
    Does not write posteriors and filtered, bias corrected images
    argument: ``--writeLess ``
defaultSuffix: (a unicode string)
    argument: ``--defaultSuffix %s``
atlasToSubjectTransform: (a boolean or a file name)
    The transform from atlas to the subject
    argument: ``--atlasToSubjectTransform %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
maxIterations: (an integer (int or long))
    Filter iterations
    argument: ``--maxIterations %d``
outputDirtyLabels: (a boolean or a file name)
    Output Dirty Label Image
    argument: ``--outputDirtyLabels %s``
outputDir: (a boolean or a directory name)
    Output directory
    argument: ``--outputDir %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

```

(continues on next page)

(continued from previous page)

```

    value.
    argument: ``--filterTimeStep %f``
implicitOutputs: (a boolean or a list of items which are a file name)
    Outputs to be made available to NiPype. Needed because not all
    BRAINSABC outputs have command line arguments.
    argument: ``--implicitOutputs %s...``
filterIteration: (an integer (int or long))
    Filter iterations
    argument: ``--filterIteration %d``
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.
    argument: ``--purePlugsThreshold %f``

```

Outputs:

```

outputDirtyLabels: (an existing file name)
    Output Dirty Label Image
outputDir: (an existing directory name)
    Output directory
outputLabels: (an existing file name)
    Output Label Image
saveState: (an existing file name)
    (optional) Filename to which save the final state of the
    registration
atlasToSubjectTransform: (an existing file name)
    The transform from atlas to the subject
atlasToSubjectInitialTransform: (an existing file name)
    The initial transform from atlas to the subject
implicitOutputs: (a list of items which are an existing file name)
    Outputs to be made available to NiPype. Needed because not all
    BRAINSABC outputs have command line arguments.
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.

```

75.20.2 BRAINSConstellationDetector**Link to code**

Wraps the executable 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 extension 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:

```

[Optional]
inputTemplateModel: (an existing file name)
    User-specified template model.,
    argument: ``--inputTemplateModel %s``
atlasLandmarks: (an existing file name)
    Atlas landmarks to be used for BRAINSFit registration
    initialization,
    argument: ``--atlasLandmarks %s``
rmpj: (a float)
    , Search radius for MPJ in unit of mm,
    argument: ``--rmpj %f``
verbose: (a boolean)
    , Show more verbose output,
    argument: ``--verbose``
atlasLandmarkWeights: (an existing file name)
    Weights associated with atlas landmarks to be used for BRAINSFit
    registration initialization,
    argument: ``--atlasLandmarkWeights %s``
inputVolume: (an existing file name)
    Input image in which to find ACPC points
    argument: ``--inputVolume %s``
otsuPercentileThreshold: (a float)
    , This is a parameter to FindLargestForegroundFilledMask, which is
    employed when acLowerBound is set and an
    outputUntransformedClippedVolume is requested.,
    argument: ``--otsuPercentileThreshold %f``
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
    argument: ``--interpolationMode %s``
rVN4: (a float)
    , Search radius for VN4 in unit of mm,
    argument: ``--rVN4 %f``
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.,
    argument: ``--rescaleIntensitiesOutputRange %s``
environ: (a dictionary with keys which are a bytes or None or a 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.,
    argument: ``--forceACPoint %s``
rac: (a float)
    , Search radius for AC in unit of mm,
    argument: ``--rac %f``
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

```

(continues on next page)

(continued from previous page)

```

components that were specified by the user on command line would be
generated. Compatible components include inputVolume, outputVolume,
outputLandmarksInInputSpace, outputLandmarksInACPCAlignedSpace, and
outputTransform.,
argument: ``--outputMRML %s``
cutOutHeadInOutputVolume: (a boolean)
, Flag to cut out just the head tissue when producing an
(un)transformed clipped volume.,
argument: ``--cutOutHeadInOutputVolume ``
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.,
argument: ``--trimRescaledIntensities %f``
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.
argument: ``--outputResampledVolume %s``
outputTransform: (a boolean or a file name)
The filename for the original space to ACPC alignment to be written
(in .h5 format).,
argument: ``--outputTransform %s``
rescaleIntensities: (a boolean)
, Flag to turn on rescaling image intensities on input.,
argument: ``--rescaleIntensities ``
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.,
argument: ``--inputLandmarksEMSP %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.,
argument: ``--outputVerificationScript %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.,
argument: ``--forceVN4Point %s``
atlasVolume: (an existing file name)
Atlas volume image to be used for BRAINSFit registration
argument: ``--atlasVolume %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.,
argument: ``--writedebuggingImagesLevel %d``
LLSModel: (an existing file name)
Linear least squares model filename in HD5 format
argument: ``--LLSModel %s``
mspQualityLevel: (an integer (int or long))
, Flag cotrols how agressive 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!.,

```

(continues on next page)

(continued from previous page)

```

        argument: ``--mspQualityLevel %d``
forceHoughEyeDetectorReportFailure: (a boolean)
    , Flag indicates whether the Hough eye detector should report
    failure,
    argument: ``--forceHoughEyeDetectorReportFailure ``
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.,
    argument: ``--outputLandmarksInInputSpace %s``
BackgroundFillValue: (a unicode string)
    Fill the background of image with specified short int value. Enter
    number or use BIGNEG for a large negative number.
    argument: ``--BackgroundFillValue %s``
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.,
    argument: ``--forcePCPoint %s``
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.,
    argument: ``--writeBranded2DImage %s``
forceRPPoint: (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.,
    argument: ``--forceRPPoint %s``
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.,
    argument: ``--outputLandmarksInACPCAlignedSpace %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.
    argument: ``--outputUntransformedClippedVolume %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.,
    argument: ``--acLowerBound %f``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
resultsDir: (a boolean or a directory name)
    , The directory for the debugging images to be written.,
    argument: ``--resultsDir %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

```

(continues on next page)

(continued from previous page)

```

    images.,
    argument: ``--houghEyeDetectorMode %d``
rpc: (a float)
    , Search radius for PC in unit of mm,
    argument: ``--rpc %f``
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.
    argument: ``--outputVolume %s``
debug: (a boolean)
    , Show internal debugging information.,
    argument: ``--debug ``

```

Outputs:

```

outputUntransformedClippedVolume: (an existing file name)
    Output image in which to store neck-clipped input image, with the
    use of --acLowerBound and maybe --cutOutHeadInUntransformedVolume.
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.,
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.
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.,
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.,
resultsDir: (an existing directory name)
    , The directory for the debugging images to be written.,
outputResampledVolume: (an existing file name)
    ACPC-aligned output image in a resampled unifor space. Currently
    this is a 1mm, 256^3, Identity direction image.
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.,
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.,
outputTransform: (an existing file name)
    The filename for the original space to ACPC alignment to be written
    (in .h5 format).,

```

75.20.3 BRAINSCreateLabelMapFromProbabilityMaps

[Link to code](#)

Wraps the executable command “ BRAINSCreateLabelMapFromProbabilityMaps “.

title: Create Label Map From Probability Maps (BRAINS)

category: Segmentation.Specialized

description: Given A list of Probability Maps, generate a LabelMap.

Inputs:

```
[Optional]
foregroundPriors: (a list of items which are an integer (int or
                  long))
    A list: For each Prior Label, 1 if foreground, 0 if background
    argument: ``--foregroundPriors %s``
dirtyLabelVolume: (a boolean or a file name)
    the labels prior to cleaning
    argument: ``--dirtyLabelVolume %s``
inputProbabilityVolume: (a list of items which are an existing file
                        name)
    The list of proobabilityimages.
    argument: ``--inputProbabilityVolume %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
    argument: ``--priorLabelCodes %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
nonAirRegionMask: (an existing file name)
    a mask representing the 'NonAirRegion' -- Just force pixels in this
    region to zero
    argument: ``--nonAirRegionMask %s``
inclusionThreshold: (a float)
    tolerance for inclusion
    argument: ``--inclusionThreshold %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
cleanLabelVolume: (a boolean or a file name)
    the foreground labels volume
    argument: ``--cleanLabelVolume %s``
```

Outputs:

```
dirtyLabelVolume: (an existing file name)
    the labels prior to cleaning
cleanLabelVolume: (an existing file name)
    the foreground labels volume
```

75.20.4 BRAINSCut

[Link to code](#)

Wraps the executable 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:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--numberOfTrees %d``
createVectors: (a boolean)
    create vectors for training neural net
    argument: ``--createVectors ``
netConfiguration: (an existing file name)
    XML File defining BRAINSCut parameters. OLD NAME. PLEASE USE
    modelConfigurationFilename instead.
    argument: ``--netConfiguration %s``
modelConfigurationFilename: (an existing file name)
    XML File defining BRAINSCut parameters
    argument: ``--modelConfigurationFilename %s``
applyModel: (a boolean)
    apply the neural net
    argument: ``--applyModel ``
multiStructureThreshold: (a boolean)
    multiStructureThreshold module to deal with overlapping area
    argument: ``--multiStructureThreshold ``
histogramEqualization: (a boolean)
    A Histogram Equalization process could be added to the
    creating/applying process from Subject To Atlas. Default is false,
    which genreate input vectors without Histogram Equalization.
    argument: ``--histogramEqualization ``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
NoTrainingVectorShuffling: (a boolean)
    If this flag is on, there will be no shuffling.
    argument: ``--NoTrainingVectorShuffling ``
trainModel: (a boolean)
    train the neural net
    argument: ``--trainModel ``
method: ('RandomForest' or 'ANN')
    argument: ``--method %s``
trainModelStartIndex: (an integer (int or long))
    Starting iteration for training
    argument: ``--trainModelStartIndex %d``
verbose: (an integer (int or long))
    print out some debugging information
    argument: ``--verbose %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 )
    argument: ``--validate ``
computeSSEOn: (a boolean)
    compute Sum of Square Error (SSE) along the trained model until the
    number of iteration given in the modelConfigurationFilename file
```

(continues on next page)

(continued from previous page)

```

        argument: ``--computeSSEOn ``
generateProbability: (a boolean)
    Generate probability map
        argument: ``--generateProbability ``
modelName: (a unicode string)
    model file name given from user (not by xml configuration file)
        argument: ``--modelName %s``
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.
        argument: ``--randomTreeDepth %d``

```

Outputs:

None

75.20.5 BRAINSMultiSTAPLE

[Link to code](#)

Wraps the executable 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:

```

[Optional]
resampledVolumePrefix: (a unicode string)
    if given, write out resampled volumes with this prefix
        argument: ``--resampledVolumePrefix %s``
outputMultiSTAPLE: (a boolean or a file name)
    the MultiSTAPLE average of input label volumes
        argument: ``--outputMultiSTAPLE %s``
inputCompositeT1Volume: (an existing file name)
    Composite T1, all label maps transofrmed into the space for this
    image.
        argument: ``--inputCompositeT1Volume %s``
args: (a unicode string)
    Additional parameters to the command
        argument: ``%s``
inputLabelVolume: (a list of items which are an existing file name)
    The list of proobabilityimages.
        argument: ``--inputLabelVolume %s...``
labelForUndecidedPixels: (an integer (int or long))
    Label for undecided pixels
        argument: ``--labelForUndecidedPixels %d``
outputConfusionMatrix: (a boolean or a file name)
    Confusion Matrix
        argument: ``--outputConfusionMatrix %s``
inputTransform: (a list of items which are an existing file name)
    transforms to apply to label volumes
        argument: ``--inputTransform %s...``
environ: (a dictionary with keys which are a 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
skipResampling: (a boolean)

```

(continues on next page)

(continued from previous page)

```
Omit resampling images into reference space
argument: ``--skipResampling``
```

Outputs:

```
outputMultiSTAPLE: (an existing file name)
    the MultiSTAPLE average of input label volumes
outputConfusionMatrix: (an existing file name)
    Confusion Matrix
```

75.20.6 BRAINSROIAuto**Link to code**

Wraps the executable 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:

```
[Optional]
outputROIMaskVolume: (a boolean or a file name)
    The ROI automatically found from the input image.
    argument: ``--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.
    argument: ``--outputVolumePixelType %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.
    argument: ``--closingSize %f``
otsuPercentileThreshold: (a float)
    Parameter to the Otsu threshold algorithm.
    argument: ``--otsuPercentileThreshold %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.
    argument: ``--ROIAutoDilateSize %f``
maskOutput: (a boolean)
```

(continues on next page)

(continued from previous page)

```

    The inputVolume multiplied by the ROI mask.
    argument: ``--maskOutput ``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
inputVolume: (an existing file name)
    The input image for finding the largest region filled mask.
    argument: ``--inputVolume %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
cropOutput: (a boolean)
    The inputVolume cropped to the region of the ROI mask.
    argument: ``--cropOutput ``
outputVolume: (a boolean or a file name)
    The inputVolume with optional [maskOutput|cropOutput] to the region
    of the brain mask.
    argument: ``--outputVolume %s``
thresholdCorrectionFactor: (a float)
    A factor to scale the Otsu algorithm's result threshold, in case
    clipping mangles the image.
    argument: ``--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.

```

75.20.7 BinaryMaskEditorBasedOnLandmarks[Link to code](#)

Wraps the executable 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:**

```

[Optional]
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.
    argument: ``--inputLandmarkNames %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)
    argument: ``--setCutDirectionForLandmark %s``

```

(continues on next page)

(continued from previous page)

```

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.
    argument: ``--inputLandmarkNamesForObliquePlane %s``
inputBinaryVolume: (an existing file name)
    Input binary image in which to be edited
    argument: ``--inputBinaryVolume %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.
    argument: ``--setCutDirectionForObliquePlane %s``
inputLandmarksFilename: (an existing file name)
    The filename for the landmark definition file in the same format
    produced by Slicer3 (.fcsv).
    argument: ``--inputLandmarksFilename %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputBinaryVolume: (a boolean or a file name)
    Output binary image in which to be edited
    argument: ``--outputBinaryVolume %s``

```

Outputs:

```

outputBinaryVolume: (an existing file name)
    Output binary image in which to be edited

```

75.20.8 ESLR[Link to code](#)

Wraps the executable 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:

```

[Optional]
inputVolume: (an existing file name)
    Input Label Volume
    argument: ``--inputVolume %s``
low: (an integer (int or long))
    The lower bound of the labels to be used.
    argument: ``--low %d``
high: (an integer (int or long))
    The higher bound of the labels to be used.
    argument: ``--high %d``
openingSize: (an integer (int or long))
    The opening size for hole filling.
    argument: ``--openingSize %d``

```

(continues on next page)

(continued from previous page)

```

preserveOutside: (a boolean)
    For values outside the specified range, preserve those values.
    argument: ``--preserveOutside``
closingSize: (an integer (int or long))
    The closing size for hole filling.
    argument: ``--closingSize %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
safetySize: (an integer (int or long))
    The safetySize size for the clipping region.
    argument: ``--safetySize %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
outputVolume: (a boolean or a file name)
    Output Label Volume
    argument: ``--outputVolume %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``

```

Outputs:

```

outputVolume: (an existing file name)
    Output Label Volume

```

75.21 interfaces.semtools.utilities.brains

75.21.1 BRAINSAlignMSP

[Link to code](#)

Wraps the executable command “BRAINSAlignMSP”.

title: Align Mid Sagittal Brain (BRAINS)

category: Utilities.BRAINS

description: Resample an image into ACPC alignment ACPCDetect

Inputs:

```

[Optional]
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
    argument: ``--interpolationMode %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.,
    argument: ``--writedebuggingImagesLevel %d``
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

```

(continues on next page)

(continued from previous page)

```

        seconds), 3=best estimate (58 seconds).,
        argument: ``--mspQualityLevel %d``
inputVolume: (an existing file name)
        , The Image to be resampled,
        argument: ``--inputVolume %s``
resultsDir: (a boolean or a directory name)
        , The directory for the results to be written.,
        argument: ``--resultsDir %s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
        Environment variables
OutputresampleMSP: (a boolean or a file name)
        , The image to be output.,
        argument: ``--OutputresampleMSP %s``
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.,
        argument: ``--rescaleIntensitiesOutputRange %s``
numberOfThreads: (an integer (int or long))
        Explicitly specify the maximum number of threads to use.
        argument: ``--numberOfThreads %d``
verbose: (a boolean)
        , Show more verbose output,
        argument: ``--verbose ``
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.,
        argument: ``--trimRescaledIntensities %f``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
rescaleIntensities: (a boolean)
        , Flag to turn on rescaling image intensities on input.,
        argument: ``--rescaleIntensities ``
BackgroundFillValue: (a unicode string)
        Fill the background of image with specified short int value. Enter
        number or use BIGNEG for a large negative number.
        argument: ``--BackgroundFillValue %s``

```

Outputs:

```

resultsDir: (an existing directory name)
        , The directory for the results to be written.,
OutputresampleMSP: (an existing file name)
        , The image to be output.,

```

75.21.2 BRAINSClipInferior[Link to code](#)

Wraps the executable 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:

```
[Optional]
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.,
    argument: ``--acLowerBound %f``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Input image to make a clipped short int copy from.
    argument: ``--inputVolume %s``
BackgroundFillValue: (a unicode string)
    Fill the background of image with specified short int value. Enter
    number or use BIGNNEG for a large negative number.
    argument: ``--BackgroundFillValue %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Output image, a short int copy of the upper portion of the input
    image, filled with BackgroundFillValue.
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Output image, a short int copy of the upper portion of the input
    image, filled with BackgroundFillValue.
```

75.21.3 BRAINSConstellationModeler

[Link to code](#)

Wraps the executable command “ BRAINSConstellationModeler “.

title: Generate Landmarks Model (BRAINS)

category: Utilities.BRAINS

description: Train up a model for BRAINSConstellationDetector

Inputs:

```
[Optional]
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.,
```

(continues on next page)

(continued from previous page)

```

        argument: ``--saveOptimizedLandmarks ``
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.,
    argument: ``--writedebuggingImagesLevel %d``
verbose: (a boolean)
    , Show more verbose output,
    argument: ``--verbose ``
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).,
    argument: ``--mspQualityLevel %d``
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.,
    argument: ``--rescaleIntensitiesOutputRange %s``
resultsDir: (a boolean or a directory name)
    , The directory for the results to be written.,
    argument: ``--resultsDir %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
inputTrainingList: (an existing file name)
    , Setup file, giving all parameters for training up a template model
    for each landmark.,
    argument: ``--inputTrainingList %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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.,
    argument: ``--trimRescaledIntensities %f``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--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.,
    argument: ``--optimizedLandmarksFilenameExtender %s``
rescaleIntensities: (a boolean)
    , Flag to turn on rescaling image intensities on input.,
    argument: ``--rescaleIntensities ``
BackgroundFillValue: (a unicode string)
    Fill the background of image with specified short int value. Enter
    number or use BIGNEG for a large negative number.

```

(continues on next page)

(continued from previous page)

```

        argument: ``--BackgroundFillValue %s``
outputModel: (a boolean or a file name)
    , The full filename of the output model file.,
    argument: ``--outputModel %s``

```

Outputs:

```

resultsDir: (an existing directory name)
    , The directory for the results to be written.,
outputModel: (an existing file name)
    , The full filename of the output model file.,

```

75.21.4 BRAINSEyeDetector[Link to code](#)

Wraps the executable command “ BRAINSEyeDetector “.

title: Eye Detector (BRAINS)

category: Utilities.BRAINS

version: 1.0

documentation-url: <http://www.nitrc.org/projects/brainscdetector/>**Inputs:**

```

[Optional]
debugDir: (a unicode string)
    A place for debug information
    argument: ``--debugDir %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    The input volume
    argument: ``--inputVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    The output volume
    argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
    The output volume

```

75.21.5 BRAINSInitializedControlPoints[Link to code](#)

Wraps the executable 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:

```
[Optional]
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.
    argument: ``--splineGridSize %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Input Volume
    argument: ``--inputVolume %s``
outputLandmarksFile: (a unicode string)
    Output filename
    argument: ``--outputLandmarksFile %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)
    argument: ``--permuteOrder %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Output Volume
    argument: ``--outputVolume %s``
```

Outputs:

```
outputVolume: (an existing file name)
    Output Volume
```

75.21.6 BRAINSLandmarkInitializer

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputFixedLandmarkFilename: (an existing file name)
    input fixed landmark. *.fcsv
```

(continues on next page)

(continued from previous page)

```

        argument: ``--inputFixedLandmarkFilename %s``
inputMovingLandmarkFilename: (an existing file name)
    input moving landmark. *.fcsv
    argument: ``--inputMovingLandmarkFilename %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--inputWeightFilename %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputTransformFilename: (a boolean or a file name)
    output transform file name (ex: ./outputTransform.mat)
    argument: ``--outputTransformFilename %s``

```

Outputs:

```

outputTransformFilename: (an existing file name)
    output transform file name (ex: ./outputTransform.mat)

```

75.21.7 BRAINSLinearModelerEPCA[Link to code](#)

Wraps the executable 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:**

```

[Optional]
inputTrainingList: (an existing file name)
    Input Training Landmark List Filename,
    argument: ``--inputTrainingList %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables

```

Outputs:

None

75.21.8 BRAINSLmkTransform

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputMovingVolume: (an existing file name)
    The filename of input moving volume
    argument: ``--inputMovingVolume %s``
inputFixedLandmarks: (an existing file name)
    Input Fixed Landmark list file in fcsv,
    argument: ``--inputFixedLandmarks %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outputResampledVolume: (a boolean or a file name)
    The filename of the output resampled volume
    argument: ``--outputResampledVolume %s``
outputAffineTransform: (a boolean or a file name)
    The filename for the estimated affine transform,
    argument: ``--outputAffineTransform %s``
inputMovingLandmarks: (an existing file name)
    Input Moving Landmark list file in fcsv,
    argument: ``--inputMovingLandmarks %s``
inputReferenceVolume: (an existing file name)
    The filename of the reference volume
    argument: ``--inputReferenceVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
outputResampledVolume: (an existing file name)
    The filename of the output resampled volume
outputAffineTransform: (an existing file name)
    The filename for the estimated affine transform,
```

75.21.9 BRAINSMush

[Link to code](#)

Wraps the executable 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:

```
[Optional]
desiredVariance: (a float)
    Desired variance within the mask for weighted sum of both images.
    argument: ``--desiredVariance %f``
lowerThresholdFactor: (a float)
    Lower threshold factor for defining the brain mask
    argument: ``--lowerThresholdFactor %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outputMask: (a boolean or a file name)
    The brain volume mask generated from the MUSH image
    argument: ``--outputMask %s``
lowerThresholdFactorPre: (a float)
    Lower threshold factor for finding an initial brain mask
    argument: ``--lowerThresholdFactorPre %f``
outputWeightsFile: (a boolean or a file name)
    Output Weights File
    argument: ``--outputWeightsFile %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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
    argument: ``--boundingBoxStart %s``
seed: (a list of items which are an integer (int or long))
    Seed Point for Brain Region Filling
    argument: ``--seed %s``
upperThresholdFactor: (a float)
    Upper threshold factor for defining the brain mask
    argument: ``--upperThresholdFactor %f``
inputSecondVolume: (an existing file name)
    Input image (2) for mixture optimization
    argument: ``--inputSecondVolume %s``
desiredMean: (a float)
    Desired mean within the mask for weighted sum of both images.
    argument: ``--desiredMean %f``
upperThresholdFactorPre: (a float)
    Upper threshold factor for finding an initial brain mask
    argument: ``--upperThresholdFactorPre %f``
inputMaskVolume: (an existing file name)
    Input label image for mixture optimization
```

(continues on next page)

(continued from previous page)

```

        argument: ``--inputMaskVolume %s``
inputFirstVolume: (an existing file name)
    Input image (1) for mixture optimization
    argument: ``--inputFirstVolume %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
    argument: ``--boundingBoxSize %s``
outputVolume: (a boolean or a file name)
    The MUSH image produced from the T1 and T2 weighted images
    argument: ``--outputVolume %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``

```

Outputs:

```

outputMask: (an existing file name)
    The brain volume mask generated from the MUSH image
outputWeightsFile: (an existing file name)
    Output Weights File
outputVolume: (an existing file name)
    The MUSH image produced from the T1 and T2 weighted images

```

75.21.10 BRAINSSnapShotWriter[Link to code](#)

Wraps the executable 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: Eunyong Regina Kim

Inputs:

```

[Optional]
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.
    argument: ``--inputVolumes %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.
    argument: ``--inputSliceToExtractInPhysicalPoint %s``
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.
    argument: ``--inputBinaryVolumes %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.
    argument: ``--inputSliceToExtractInIndex %s``
args: (a unicode string)

```

(continues on next page)

(continued from previous page)

```

Additional parameters to the command
argument: ``%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
    argument: ``--inputSliceToExtractInPercent %s``
outputFilename: (a boolean or a file name)
    2D file name of input images. Required.
    argument: ``--outputFilename %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.
    argument: ``--inputPlaneDirection %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
Environment variables

```

Outputs:

```

outputFilename: (an existing file name)
    2D file name of input images. Required.

```

75.21.11 BRAINSTransformConvert[Link to code](#)

Wraps the executable 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:

```

[Optional]
outputPrecisionType: ('double' or 'float')
    Precision type of the output transform. It can be either single
    precision or double precision
    argument: ``--outputPrecisionType %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
displacementVolume: (a boolean or a file name)
    argument: ``--displacementVolume %s``
referenceVolume: (an existing file name)
    argument: ``--referenceVolume %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
    argument: ``--outputTransformType %s``
inputTransform: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    argument: ``--inputTransform %s``
environ: (a dictionary with keys which are a bytes or None or a value
        of class 'str' and with values which are a bytes or None or a
        value of class 'str', nipy default value: {})
    Environment variables
outputTransform: (a boolean or a file name)
    argument: ``--outputTransform %s``

```

Outputs:

```

displacementVolume: (an existing file name)
outputTransform: (an existing file name)

```

75.21.12 BRAINSTrimForegroundInDirection[Link to code](#)

Wraps the executable 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:**

```

[Optional]
directionCode: (an integer (int or long))
    , This flag chooses which dimension to compare. The sign lets you
    flip direction.,
    argument: ``--directionCode %d``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
otsuPercentileThreshold: (a float)
    , This is a parameter to FindLargestForegroundFilledMask, which is
    employed to trim off air-filling noise.,
    argument: ``--otsuPercentileThreshold %f``
inputVolume: (an existing file name)
    Input image to trim off the neck (and also air-filling noise.)
    argument: ``--inputVolume %s``
BackgroundFillValue: (a unicode string)
    Fill the background of image with specified short int value. Enter
    number or use BIGNEG for a large negative number.
    argument: ``--BackgroundFillValue %s``
closingSize: (an integer (int or long))
    , This is a parameter to FindLargestForegroundFilledMask,
    argument: ``--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', nipy default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Output image with neck and air-filling noise trimmed isotropic image
    with AC at center of image.

```

(continues on next page)

(continued from previous page)

```

argument: ``--outputVolume %s``
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.,
argument: ``--headSizeLimit %f``

```

Outputs:

```

outputVolume: (an existing file name)
    Output image with neck and air-filling noise trimmed isotropic image
    with AC at center of image.

```

75.21.13 CleanUpOverlapLabels[Link to code](#)

Wraps the executable 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:

```

[Optional]
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.
    argument: ``--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
    argument: ``--outputBinaryVolumes %s...``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

```

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

```

75.21.14 FindCenterOfBrain[Link to code](#)

Wraps the executable 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:

```
[Optional]
debugTrimmedImage: (a boolean or a file name)
    argument: ``--debugTrimmedImage %s``
generateDebugImages: (a boolean)
    argument: ``--generateDebugImages ``
axis: (an integer (int or long))
    argument: ``--axis %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
debugClippedImageMask: (a boolean or a file name)
    argument: ``--debugClippedImageMask %s``
inputVolume: (an existing file name)
    The image in which to find the center.
    argument: ``--inputVolume %s``
otsuPercentileThreshold: (a float)
    argument: ``--otsuPercentileThreshold %f``
debugAfterGridComputationsForegroundImage: (a boolean or a file name)
    argument: ``--debugAfterGridComputationsForegroundImage %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
debugDistanceImage: (a boolean or a file name)
    argument: ``--debugDistanceImage %s``
headSizeLimit: (a float)
    argument: ``--headSizeLimit %f``
debugGridImage: (a boolean or a file name)
    argument: ``--debugGridImage %s``
closingSize: (an integer (int or long))
    argument: ``--closingSize %d``
backgroundValue: (an integer (int or long))
    argument: ``--backgroundValue %d``
imageMask: (an existing file name)
    argument: ``--imageMask %s``
clippedImageMask: (a boolean or a file name)
    argument: ``--clippedImageMask %s``
maximize: (a boolean)
    argument: ``--maximize ``
headSizeEstimate: (a float)
    argument: ``--headSizeEstimate %f``
```

Outputs:

```
debugTrimmedImage: (an existing file name)
debugGridImage: (an existing file name)
debugAfterGridComputationsForegroundImage: (an existing file name)
```

(continues on next page)

(continued from previous page)

```
debugClippedImageMask: (an existing file name)
clippedImageMask: (an existing file name)
debugDistanceImage: (an existing file name)
```

75.21.15 GenerateLabelMapFromProbabilityMap

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputVolumes: (a list of items which are an existing file name)
    The Input probaiblity images to be computed for lable maps
    argument: ``--inputVolumes %s...``
outputLabelVolume: (a boolean or a file name)
    The Input binary image for region of interest
    argument: ``--outputLabelVolume %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
outputLabelVolume: (an existing file name)
    The Input binary image for region of interest
```

75.21.16 ImageRegionPlotter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
useIntensityForHistogram: (a boolean)
    Create Intensity Joint Histogram instead of Quantile Joint
    Histogram
    argument: ``--useIntensityForHistogram ``
```

(continues on next page)

(continued from previous page)

```

numberOfHistogramBins: (an integer (int or long))
    the number of histogram levels
    argument: ``--numberOfHistogramBins %d``
useROIAUTO: (a boolean)
    Use ROIAUTO to compute region of interest. This cannot be used with
    inputLabelVolume
    argument: ``--useROIAUTO ``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    print debugging information,
    argument: ``--verbose ``
inputLabelVolume: (an existing file name)
    The Label Image
    argument: ``--inputLabelVolume %s``
inputVolume2: (an existing file name)
    The Input image to be computed for statistics
    argument: ``--inputVolume2 %s``
inputVolume1: (an existing file name)
    The Input image to be computed for statistics
    argument: ``--inputVolume1 %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputJointHistogramData: (a unicode string)
    output data file name
    argument: ``--outputJointHistogramData %s``
inputBinaryROIVolume: (an existing file name)
    The Input binary image for region of interest
    argument: ``--inputBinaryROIVolume %s``

```

Outputs:**None****75.21.17 JointHistogram**[Link to code](#)

Wraps the executable 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:**

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
verbose: (a boolean)
    print debugging information,
    argument: ``--verbose ``
inputMaskVolumeInXAxis: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

    Input mask volume for inputVolumeInXAxis. Histogram will be computed
    just for the masked region
    argument: ``--inputMaskVolumeInXAxis %s``
inputVolumeInXAxis: (an existing file name)
    The Input image to be computed for statistics
    argument: ``--inputVolumeInXAxis %s``
inputVolumeInYAxis: (an existing file name)
    The Input image to be computed for statistics
    argument: ``--inputVolumeInYAxis %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
inputMaskVolumeInYAxis: (an existing file name)
    Input mask volume for inputVolumeInYAxis. Histogram will be computed
    just for the masked region
    argument: ``--inputMaskVolumeInYAxis %s``
outputJointHistogramImage: (a unicode string)
    output joint histogram image file name. Histogram is usually 2D
    image.
    argument: ``--outputJointHistogramImage %s``

```

Outputs:**None****75.21.18 ShuffleVectorsModule**[Link to code](#)

Wraps the executable 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:

```

[Optional]
outputVectorFileName: (a boolean or a file name)
    output vector file name prefix. Usually end with .txt and header
    file has prost fix of .txt.hdr
    argument: ``--outputVectorFileName %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.
    argument: ``--resampleProportion %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
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVectorFileName: (an existing file name)
    input vector file name prefix. Usually end with .txt and header file

```

(continues on next page)

(continued from previous page)

```
has prost fix of .txt.hdr
argument: ``--inputVectorFileName %s``
```

Outputs:

```
outputVectorFileName: (an existing file name)
    output vector file name prefix. Usually end with .txt and header
    file has prost fix of .txt.hdr
```

75.21.19 fcsv_to_hdf5[Link to code](#)

Wraps the executable 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:

```
[Optional]
landmarkGlobPattern: (a unicode string)
    Glob pattern to select fcsv files
    argument: ``--landmarkGlobPattern %s``
versionID: (a unicode string)
    , Current version ID. It should be match with the version of BCD
    that will be using the output model file,
    argument: ``--versionID %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
landmarkTypesList: (an existing file name)
    , file containing list of landmark types,
    argument: ``--landmarkTypesList %s``
modelFile: (a boolean or a file name)
    , name of HDF5 file containing BRAINSConstellationDetector Model
    file (LLSMatrices, LLSMeans and LLSSearchRadii),
    argument: ``--modelFile %s``
landmarksInformationFile: (a boolean or a file name)
    , name of HDF5 file to write matrices into,
    argument: ``--landmarksInformationFile %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
modelFile: (an existing file name)
    , name of HDF5 file containing BRAINSConstellationDetector Model
    file (LLSMatrices, LLSMeans and LLSSearchRadii),
landmarksInformationFile: (an existing file name)
    , name of HDF5 file to write matrices into,
```

75.21.20 insertMidACPCpoint

[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputLandmarkFile: (a boolean or a file name)
    Output landmark file (.fcsv)
    argument: ``--outputLandmarkFile %s``
inputLandmarkFile: (an existing file name)
    Input landmark file (.fcsv)
    argument: ``--inputLandmarkFile %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

```
outputLandmarkFile: (an existing file name)
    Output landmark file (.fcsv)
```

75.21.21 landmarksConstellationAligner

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputLandmarksPaired: (an existing file name)
    Input landmark file (.fcsv)
    argument: ``--inputLandmarksPaired %s``
outputLandmarksPaired: (a boolean or a file name)
    Output landmark file (.fcsv)
    argument: ``--outputLandmarksPaired %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

```
outputLandmarksPaired: (an existing file name)
    Output landmark file (.fcsv)
```

75.21.22 landmarksConstellationWeights

[Link to code](#)

Wraps the executable command “ landmarksConstellationWeights “.

title: Generate Landmarks Weights (BRAINS)

category: Utilities.BRAINS

description: Train up a list of Weights for the Landmarks in BRAINSConstellationDetector

Inputs:

```
[Optional]
inputTemplateModel: (an existing file name)
    User-specified template model.,
    argument: ``--inputTemplateModel %s``
LLSModel: (an existing file name)
    Linear least squares model filename in HD5 format
    argument: ``--LLSModel %s``
inputTrainingList: (an existing file name)
    , Setup file, giving all parameters for training up a Weight list
    for landmark.,
    argument: ``--inputTrainingList %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
outputWeightsList: (a boolean or a file name)
    , The filename of a csv file which is a list of landmarks and their
    corresponding weights.,
    argument: ``--outputWeightsList %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:

```
outputWeightsList: (an existing file name)
    , The filename of a csv file which is a list of landmarks and their
    corresponding weights.,
```

76.1 interfaces.slicer.base

76.1.1 SlicerCommandLine

[Link to code](#)

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
```

Outputs:

None

76.2 interfaces.slicer.converters

76.2.1 DicomToNrrdConverter

[Link to code](#)

Wraps the executable command “DicomToNrrdConverter”.

title: 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:

```
[Optional]
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.
    argument: ``--writeProtocolGradientsFile ``
useBMatrixGradientDirections: (a boolean)
    Fill the nhdr header with the gradient directions and bvalues
    computed out of the BMatrix. Only changes behavior for Siemens data.
    argument: ``--useBMatrixGradientDirections ``
outputVolume: (a unicode string)
    Output filename (.nhdr or .nrrd)
    argument: ``--outputVolume %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputDicomDirectory: (an existing directory name)
    Directory holding Dicom series
    argument: ``--inputDicomDirectory %s``
useIdentityMeaseurementFrame: (a boolean)
    Adjust all the gradients so that the measurement frame is an
    identity matrix.
    argument: ``--useIdentityMeaseurementFrame ``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
outputDirectory: (a boolean or a directory name)
    Directory holding the output NRRD format
    argument: ``--outputDirectory %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.
    argument: ``--smallGradientThreshold %f``
```

Outputs:

```
outputDirectory: (an existing directory name)
    Directory holding the output NRRD format
```

76.2.2 OrientScalarVolume

[Link to code](#)

Wraps the executable 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:

```
[Optional]
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
  argument: ``--orientation %s``
inputVolume1: (an existing file name)
  Input volume 1
  argument: ``%s``, position: -2
args: (a unicode string)
  Additional parameters to the command
  argument: ``%s``
outputVolume: (a boolean or a file name)
  The oriented volume
  argument: ``%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
```

Outputs:

```
outputVolume: (an existing file name)
  The oriented volume
```

76.3 interfaces.slicer.diffusion.diffusion

76.3.1 DTIexport

[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputFile: (a boolean or a file name)
    Output DTI file
    argument: ``%s``, position: -1
inputTensor: (an existing file name)
    Input DTI volume
    argument: ``%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
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

```
outputFile: (an existing file name)
    Output DTI file
```

76.3.2 DTIimport

[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputTensor: (a boolean or a file name)
    Output DTI volume
    argument: ``%s``, position: -1
inputFile: (an existing file name)
    Input DTI file
    argument: ``%s``, position: -2
testingmode: (a boolean)
    Enable testing mode. Sample helix file (helix-DTI.nhdr) will be
    loaded into Slicer and converted in Nifti.
    argument: ``--testingmode``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
```

Outputs:


```
outputTensor: (an existing file name)
               Output DTI volume
```

76.3.3 DWIJointRicianLMMSEFilter

[Link to code](#)

Wraps the executable command “DWIJointRicianLMMSEFilter”.

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:

```
[Optional]
compressOutput: (a boolean)
                 Compress the data of the compressed file using gzip
                 argument: ``--compressOutput ``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
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).
    argument: ``--ng %d``
rf: (a list of items which are an integer (int or long))
    Filtering radius.
    argument: ``--rf %s``
inputVolume: (an existing file name)
             Input DWI volume.
             argument: ``%s``, position: -2
outputVolume: (a boolean or a file name)
              Output DWI volume.
              argument: ``%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
re: (a list of items which are an integer (int or long))
    Estimation radius.
    argument: ``--re %s``
```

Outputs:

```
outputVolume: (an existing file name)
               Output DWI volume.
```

76.3.4 DWIRicianLMMSEFilter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
rf: (a list of items which are an integer (int or long))
    Filtering radius.
    argument: ``--rf %s``
inputVolume: (an existing file name)
              Input DWI volume.
              argument: ``%s``, position: -2
mnvf: (an integer (int or long))
      Minimum number of voxels in kernel used for filtering.
      argument: ``--mnvf %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
re: (a list of items which are an integer (int or long))
    Estimation radius.
    argument: ``--re %s``
iter: (an integer (int or long))
      Number of iterations for the noise removal filter.
      argument: ``--iter %d``
uav: (a boolean)
      Use absolute value in case of negative square.
      argument: ``--uav ``
maxnstd: (an integer (int or long))
          Maximum allowed noise standard deviation.
          argument: ``--maxnstd %d``
```

(continues on next page)

(continued from previous page)

```

hrf: (a float)
    How many histogram bins per unit interval.
    argument: ``--hrf %f``
compressOutput: (a boolean)
    Compress the data of the compressed file using gzip
    argument: ``--compressOutput``
mnve: (an integer (int or long))
    Minimum number of voxels in kernel used for estimation.
    argument: ``--mnve %d``
outputVolume: (a boolean or a file name)
    Output DWI volume.
    argument: ``%s``, position: -1
minnstd: (an integer (int or long))
    Minimum allowed noise standard deviation.
    argument: ``--minnstd %d``

```

Outputs:

```

outputVolume: (an existing file name)
    Output DWI volume.

```

76.3.5 DWIToDTIEstimation[Link to code](#)

Wraps the executable 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, weighted 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:

```

[Optional]
mask: (an existing file name)
    Mask where the tensors will be computed
    argument: ``--mask %s``
shiftNeg: (a boolean)
    Shift eigenvalues so all are positive (accounts for bad tensors
    related to noise or acquisition error)
    argument: ``--shiftNeg``
outputTensor: (a boolean or a file name)
    Estimated DTI volume
    argument: ``%s``, position: -2
enumeration: ('LS' or 'WLS')
    LS: Least Squares, WLS: Weighted Least Squares

```

(continues on next page)

(continued from previous page)

```

        argument: ``--enumeration %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Input DWI volume
    argument: ``%s``, 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', nipype default value: {})
    Environment variables
outputBaseline: (a boolean or a file name)
    Estimated baseline volume
    argument: ``%s``, position: -1

```

Outputs:

```

outputTensor: (an existing file name)
    Estimated DTI volume
outputBaseline: (an existing file name)
    Estimated baseline volume

```

76.3.6 DiffusionTensorScalarMeasurements[Link to code](#)

Wraps the executable 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:

```

[Optional]
inputVolume: (an existing file name)
    Input DTI volume
    argument: ``%s``, position: -3
outputScalar: (a boolean or a file name)
    Scalar volume derived from tensor
    argument: ``%s``, position: -1
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

```

(continues on next page)

(continued from previous page)

```

    argument: ``--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
args: (a unicode string)
    Additional parameters to the command
argument: ``%s``

```

Outputs:

```

outputScalar: (an existing file name)
    Scalar volume derived from tensor

```

76.3.7 DiffusionWeightedVolumeMasking[Link to code](#)

Wraps the executable command “DiffusionWeightedVolumeMasking”.

title: Diffusion Weighted Volume Masking

category: Diffusion.Diffusion Weighted Images

description: <p>Performs a mask calculation from a diffusion weighted (DW) image.</p><p>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.</p>

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:

```

[Optional]
otsuomegathreshold: (a float)
    Control the sharpness of the threshold in the Otsu computation. 0:
    lower threshold, 1: higher threshold
    argument: ``--otsuomegathreshold %f``
thresholdMask: (a boolean or a file name)
    Otsu Threshold Mask
    argument: ``%s``, position: -1
removeislands: (a boolean)
    Remove Islands in Threshold Mask?
    argument: ``--removeislands``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    Input DWI volume
    argument: ``%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
outputBaseline: (a boolean or a file name)
    Estimated baseline volume
    argument: ``%s``, position: -2

```

Outputs:

```
thresholdMask: (an existing file name)
    Otsu Threshold Mask
outputBaseline: (an existing file name)
    Estimated baseline volume
```

76.3.8 ResampleDTIVolume

[Link to code](#)

Wraps the executable 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:

```
[Optional]
Reference: (an existing file name)
    Reference Volume (spacing, size, orientation, origin)
    argument: ``--Reference %s``
correction: ('zero' or 'none' or 'abs' or 'nearest')
    Correct the tensors if computed tensor is not semi-definite positive
    argument: ``--correction %s``
inputVolume: (an existing file name)
    Input volume to be resampled
    argument: ``%s``, position: -2
notbulk: (a boolean)
    The transform following the BSpline transform is not set as a bulk
    transform for the BSpline transform
    argument: ``--notbulk``
Inverse_ITK_Transformation: (a boolean)
    Inverse the transformation before applying it from output image to
    input image (only for rigid and affine transforms)
    argument: ``--Inverse_ITK_Transformation``
interpolation: ('linear' or 'nn' or 'ws' or 'bs')
    Sampling algorithm (linear , nn (nearest neighborhood), ws
    (WindowedSinc), bs (BSpline) )
    argument: ``--interpolation %s``
environ: (a dictionary with keys which are a bytes or None or a 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
    argument: ``--hfieldtype %s``
```

(continues on next page)

(continued from previous page)

```

window_function: ('h' or 'c' or 'w' or 'l' or 'b')
    Window Function , h = Hamming , c = Cosine , w = Welch , l = Lanczos
    , b = Blackman
    argument: ``--window_function %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)
    argument: ``--transform_tensor_method %s``
defField: (an existing file name)
    File containing the deformation field (3D vector image containing
    vectors with 3 components)
    argument: ``--defField %s``
image_center: ('input' or 'output')
    Image to use to center the transform (used only if 'Centered
    Transform' is selected)
    argument: ``--image_center %s``
centered_transform: (a boolean)
    Set the center of the transformation to the center of the input
    image (only for rigid and affine transforms)
    argument: ``--centered_transform ``
number_of_thread: (an integer (int or long))
    Number of thread used to compute the output image
    argument: ``--number_of_thread %d``
spacing: (a list of items which are a float)
    Spacing along each dimension (0 means use input spacing)
    argument: ``--spacing %s``
transform_matrix: (a list of items which are a float)
    12 parameters of the transform matrix by rows ( --last 3 being
    translation-- )
    argument: ``--transform_matrix %s``
rotation_point: (a list of items which are any value)
    Center of rotation (only for rigid and affine transforms)
    argument: ``--rotation_point %s``
transformationFile: (an existing file name)
    argument: ``--transformationFile %s``
transform: ('rt' or 'a')
    Transform algorithm, rt = Rigid Transform, a = Affine Transform
    argument: ``--transform %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)
    argument: ``--spaceChange ``
origin: (a list of items which are any value)
    Origin of the output Image
    argument: ``--origin %s``
spline_order: (an integer (int or long))
    Spline Order (Spline order may be from 0 to 5)
    argument: ``--spline_order %d``
size: (a list of items which are a float)
    Size along each dimension (0 means use input size)
    argument: ``--size %s``
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)
    argument: ``--direction_matrix %s``

```

(continues on next page)

(continued from previous page)

```

default_pixel_value: (a float)
    Default pixel value for samples falling outside of the input region
    argument: ``--default_pixel_value %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
transform_order: ('input-to-output' or 'output-to-input')
    Select in what order the transforms are read
    argument: ``--transform_order %s``
outputVolume: (a boolean or a file name)
    Resampled Volume
    argument: ``%s``, position: -1

```

Outputs:

```

outputVolume: (an existing file name)
    Resampled Volume

```

76.3.9 TractographyLabelMapSeeding[Link to code](#)

Wraps the executable 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:

```

[Optional]
seedspace: (a float)
    Spacing (in mm) between seed points, only matters if use Use Index
    Space is off
    argument: ``--seedspace %f``
writetofile: (a boolean)
    Write fibers to disk or create in the scene?
    argument: ``--writetofile``
maximumlength: (a float)
    Maximum length of fibers (in mm)
    argument: ``--maximumlength %f``
label: (an integer (int or long))
    Label value that defines seeding region.
    argument: ``--label %d``
integrationsteplength: (a float)
    Distance between points on the same fiber in mm
    argument: ``--integrationsteplength %f``
InputVolume: (an existing file name)
    Input DTI volume
    argument: ``%s``, position: -2
stoppingvalue: (a float)

```

(continues on next page)

(continued from previous page)

```

    Tractography will stop when the stopping measurement drops below
    this value
    argument: ``--stoppingvalue %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', nipyte default value: {})
    Environment variables
clthreshold: (a float)
    Minimum Linear Measure for the seeding to start.
    argument: ``--clthreshold %f``
OutputFibers: (a boolean or a file name)
    Tractography result
    argument: ``%s``, position: -1
stoppingcurvature: (a float)
    Tractography will stop if radius of curvature becomes smaller than
    this number units are degrees per mm
    argument: ``--stoppingcurvature %f``
outputdirectory: (a boolean or a directory name)
    Directory in which to save fiber(s)
    argument: ``--outputdirectory %s``
minimumlength: (a float)
    Minimum length of the fibers (in mm)
    argument: ``--minimumlength %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
useindexspace: (a boolean)
    Seed at IJK voxel grid
    argument: ``--useindexspace ``
randomgrid: (a boolean)
    Enable random placing of seeds
    argument: ``--randomgrid ``
stoppingmode: ('LinearMeasure' or 'FractionalAnisotropy')
    Tensor measurement used to stop the tractography
    argument: ``--stoppingmode %s``
inputroi: (an existing file name)
    Label map with seeding ROIs
    argument: ``--inputroi %s``
name: (a unicode string)
    Name to use for fiber files
    argument: ``--name %s``

```

Outputs:

```

OutputFibers: (an existing file name)
    Tractography result
outputdirectory: (an existing directory name)
    Directory in which to save fiber(s)

```

76.4 interfaces.slicer.filtering.arithmetic

76.4.1 AddScalarVolumes

[Link to code](#)

Wraps the executable 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:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume2: (an existing file name)
    Input volume 2
    argument: ``%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.
    argument: ``--order %s``
inputVolume1: (an existing file name)
    Input volume 1
    argument: ``%s``, 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
outputVolume: (a boolean or a file name)
    Volume1 + Volume2
    argument: ``%s``, position: -1
```

Outputs:

```
outputVolume: (an existing file name)
    Volume1 + Volume2
```

76.4.2 CastScalarVolume

[Link to code](#)

Wraps the executable 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:

```
[Optional]
InputVolume: (an existing file name)
    Input volume, the volume to cast.
    argument: ``%s``, position: -2
```

(continues on next page)

(continued from previous page)

```

type: ('Char' or 'UnsignedChar' or 'Short' or 'UnsignedShort' or
      'Int' or 'UnsignedInt' or 'Float' or 'Double')
      Type for the new output volume.
      argument: ``--type %s``
OutputVolume: (a boolean or a file name)
      Output volume, cast to the new type.
      argument: ``%s``, position: -1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a 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

```

Outputs:

```

OutputVolume: (an existing file name)
      Output volume, cast to the new type.

```

76.4.3 MaskScalarVolume[Link to code](#)

Wraps the executable 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:

```

[Optional]
label: (an integer (int or long))
      Label value in the Mask Volume to use as the mask
      argument: ``--label %d``
OutputVolume: (a boolean or a file name)
      Output volume: Input Volume masked by label value from Mask Volume
      argument: ``%s``, position: -1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
MaskVolume: (an existing file name)
      Label volume containing the mask
      argument: ``%s``, position: -2
replace: (an integer (int or long))
      Value to use for the output volume outside of the mask
      argument: ``--replace %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: {})

```

(continues on next page)

(continued from previous page)

```

Environment variables
InputVolume: (an existing file name)
Input volume to be masked
argument: ``%s``, position: -3

```

Outputs:

```

OutputVolume: (an existing file name)
Output volume: Input Volume masked by label value from Mask Volume

```

76.4.4 MultiplyScalarVolumes

[Link to code](#)

Wraps the executable 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:

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume2: (an existing file name)
    Input volume 2
    argument: ``%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.
    argument: ``--order %s``
inputVolume1: (an existing file name)
    Input volume 1
    argument: ``%s``, 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
outputVolume: (a boolean or a file name)
    Volume1 * Volume2
    argument: ``%s``, position: -1

```

Outputs:

```

outputVolume: (an existing file name)
    Volume1 * Volume2

```

76.4.5 SubtractScalarVolumes

[Link to code](#)

Wraps the executable 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:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume2: (an existing file name)
    Input volume 2
    argument: ``%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.
    argument: ``--order %s``
inputVolume1: (an existing file name)
    Input volume 1
    argument: ``%s``, 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
outputVolume: (a boolean or a file name)
    Volume1 - Volume2
    argument: ``%s``, position: -1
```

Outputs:

```
outputVolume: (an existing file name)
    Volume1 - Volume2
```

76.5 interfaces.slicer.filtering.checkerboardfilter

76.5.1 CheckerBoardFilter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume2: (an existing file name)
    Second Input volume
    argument: ``%s``, position: -2
inputVolume1: (an existing file name)
    First Input volume
    argument: ``%s``, position: -3
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.
    argument: ``--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', nipype default value: {})
    Environment variables
outputVolume: (a boolean or a file name)
    Output filtered
    argument: ``%s``, position: -1
```

Outputs:

```
outputVolume: (an existing file name)
    Output filtered
```

76.6 interfaces.slicer.filtering.denoising

76.6.1 CurvatureAnisotropicDiffusion

Link to code

Wraps the executable 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:

```
[Optional]
inputVolume: (an existing file name)
    Input volume to be filtered
    argument: ``%s``, position: -2
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.
    argument: ``--timeStep %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--conductance %f``
outputVolume: (a boolean or a file name)
    Output filtered
    argument: ``%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
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.
    argument: ``--iterations %d``
```

Outputs:

```
outputVolume: (an existing file name)
    Output filtered
```

76.6.2 GaussianBlurImageFilter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputVolume: (an existing file name)
    Input volume
```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``, position: -2
outputVolume: (a boolean or a file name)
    Blurred Volume
        argument: ``%s``, position: -1
sigma: (a float)
    Sigma value in physical units (e.g., mm) of the Gaussian kernel
        argument: ``--sigma %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
args: (a unicode string)
    Additional parameters to the command
        argument: ``%s``

```

Outputs:

```

outputVolume: (an existing file name)
    Blurred Volume

```

76.6.3 GradientAnisotropicDiffusion[Link to code](#)

Wraps the executable 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/GradientAnisotropicDiffusion>

contributor: Bill Lorensen (GE)

acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software Consortium

Inputs:

```

[Optional]
inputVolume: (an existing file name)
    Input volume to be filtered
        argument: ``%s``, position: -2
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.
        argument: ``--timeStep %f``
args: (a unicode string)
    Additional parameters to the command
        argument: ``%s``
conductance: (a float)

```

(continues on next page)

(continued from previous page)

```

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.
argument: ``--conductance %f``
outputVolume: (a boolean or a file name)
Output filtered
argument: ``%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
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.
argument: ``--iterations %d``

```

Outputs:

```

outputVolume: (an existing file name)
Output filtered

```

76.6.4 MedianImageFilter[Link to code](#)

Wraps the executable 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:

```

[Optional]
inputVolume: (an existing file name)
Input volume to be filtered
argument: ``%s``, position: -2
outputVolume: (a boolean or a file name)
Output filtered
argument: ``%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
args: (a unicode string)
Additional parameters to the command

```

(continues on next page)

(continued from previous page)

```

    argument: ``%s``
neighborhood: (a list of items which are an integer (int or long))
    The size of the neighborhood in each dimension
    argument: ``--neighborhood %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Output filtered

```

76.7 interfaces.slicer.filtering.extractskelton

76.7.1 ExtractSkeleton

[Link to code](#)

Wraps the executable 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:

```

[Optional]
dontPrune: (a boolean)
    Return the full skeleton, not just the maximal skeleton
    argument: ``--dontPrune``
InputImageFileName: (an existing file name)
    Input image
    argument: ``%s``, position: -2
numPoints: (an integer (int or long))
    Number of points used to represent the skeleton
    argument: ``--numPoints %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
pointsFile: (a unicode string)
    Name of the file to store the coordinates of the central (1D)
    skeleton points
    argument: ``--pointsFile %s``
OutputImageFileName: (a boolean or a file name)
    Skeleton of the input image
    argument: ``%s``, position: -1
type: ('1D' or '2D')
    Type of skeleton to create
    argument: ``--type %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a

```

(continues on next page)

(continued from previous page)

```
value of class 'str', nipy default value: {}
Environment variables
```

Outputs:

```
OutputImageFileName: (an existing file name)
Skeleton of the input image
```

76.8 interfaces.slicer.filtering.histogrammatching

76.8.1 HistogramMatching

[Link to code](#)

Wraps the executable 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 same MR 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:

```
[Optional]
numberOfHistogramLevels: (an integer (int or long))
    The number of histogram levels to use
    argument: ``--numberOfHistogramLevels %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
threshold: (a boolean)
    If on, only pixels above the mean in each volume are thresholded.
    argument: ``--threshold``
inputVolume: (an existing file name)
    Input volume to be filtered
    argument: ``%s``, position: -3
referenceVolume: (an existing file name)
    Input volume whose histogram will be matched
    argument: ``%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
```

(continues on next page)

(continued from previous page)

```

outputVolume: (a boolean or a file name)
    Output volume. This is the input volume with intensities matched to
    the reference volume.
    argument: ``%s``, position: -1
numberOfMatchPoints: (an integer (int or long))
    The number of match points to use
    argument: ``--numberOfMatchPoints %d``

```

Outputs:

```

outputVolume: (an existing file name)
    Output volume. This is the input volume with intensities matched to
    the reference volume.

```

76.9 interfaces.slicer.filtering.imagelabelcombine

76.9.1 ImageLabelCombine

[Link to code](#)

Wraps the executable 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/ImageLabelCombine>

contributor: Alex Yarmarkovich (SPL, BWH)

Inputs:

```

[Optional]
InputLabelMap_B: (an existing file name)
    Label map image
    argument: ``%s``, position: -2
OutputLabelMap: (a boolean or a file name)
    Resulting Label map image
    argument: ``%s``, position: -1
first_overwrites: (a boolean)
    Use first or second label when both are present
    argument: ``--first_overwrites``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
InputLabelMap_A: (an existing file name)
    Label map image
    argument: ``%s``, 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

```

Outputs:

```

OutputLabelMap: (an existing file name)
    Resulting Label map image

```

76.10 interfaces.slicer.filtering.morphology

76.10.1 GrayscaleFillHoleImageFilter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputVolume: (an existing file name)
    Input volume to be filtered
    argument: ``%s``, position: -2
outputVolume: (a boolean or a file name)
    Output filtered
    argument: ``%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
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

```
outputVolume: (an existing file name)
    Output filtered
```

76.10.2 GrayscaleGrindPeakImageFilter

[Link to code](#)

Wraps the executable 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:

```
[Optional]
inputVolume: (an existing file name)
    Input volume to be filtered
    argument: ``%s``, position: -2
outputVolume: (a boolean or a file name)
    Output filtered
    argument: ``%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
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
```

Outputs:

```
outputVolume: (an existing file name)
    Output filtered
```

76.11 interfaces.slicer.filtering.n4itkbiasfieldcorrection

76.11.1 N4ITKBiasFieldCorrection

[Link to code](#)

Wraps the executable 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:

```
[Optional]
outputbiasfield: (a boolean or a file name)
    Recovered bias field (OPTIONAL)
    argument: ``--outputbiasfield %s``
convergencythreshold: (a float)
    Stopping criterion for the iterative bias estimation. Larger values
    will lead to smaller execution time.
    argument: ``--convergencythreshold %f``
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.
    argument: ``--shrinkfactor %d``
inputimage: (an existing file name)
    Input image where you observe signal inhomogeneity
    argument: ``--inputimage %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.
    argument: ``--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)
    argument: ``--meshresolution %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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.
    argument: ``--splinedistance %f``
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.
    argument: ``--histogramsharpening %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
```

(continues on next page)

(continued from previous page)

```

    result.
    argument: ``--bsplineorder %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
weightimage: (an existing file name)
    Weight Image
    argument: ``--weightimage %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.
    argument: ``--iterations %s``
outputimage: (a boolean or a file name)
    Result of processing
    argument: ``--outputimage %s``

```

Outputs:

```

outputbiasfield: (an existing file name)
    Recovered bias field (OPTIONAL)
outputimage: (an existing file name)
    Result of processing

```

76.12 interfaces.slicer.filtering.resamplescalarvectordwivolume

76.12.1 ResampleScalarVectorDWIVolume

Link to code

Wraps the executable 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:

```

[Optional]
Reference: (an existing file name)
    Reference Volume (spacing, size, orientation, origin)
    argument: ``--Reference %s``
inputVolume: (an existing file name)
    Input Volume to be resampled

```

(continues on next page)

(continued from previous page)

```

        argument: ``%s``, position: -2
notbulk: (a boolean)
    The transform following the BSpline transform is not set as a bulk
    transform for the BSpline transform
    argument: ``--notbulk``
Inverse_ITK_Transformation: (a boolean)
    Inverse the transformation before applying it from output image to
    input image
    argument: ``--Inverse_ITK_Transformation``
interpolation: ('linear' or 'nn' or 'ws' or 'bs')
    Sampling algorithm (linear or nn (nearest neighborhood), ws
    (WindowedSinc), bs (BSpline) )
    argument: ``--interpolation %s``
environ: (a dictionary with keys which are a bytes or None or a 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
    argument: ``--hfieldtype %s``
window_function: ('h' or 'c' or 'w' or 'l' or 'b')
    Window Function , h = Hamming , c = Cosine , w = Welch , l = Lanczos
    , b = Blackman
    argument: ``--window_function %s``
defField: (an existing file name)
    File containing the deformation field (3D vector image containing
    vectors with 3 components)
    argument: ``--defField %s``
image_center: ('input' or 'output')
    Image to use to center the transform (used only if 'Centered
    Transform' is selected)
    argument: ``--image_center %s``
centered_transform: (a boolean)
    Set the center of the transformation to the center of the input
    image
    argument: ``--centered_transform``
number_of_thread: (an integer (int or long))
    Number of thread used to compute the output image
    argument: ``--number_of_thread %d``
spacing: (a list of items which are a float)
    Spacing along each dimension (0 means use input spacing)
    argument: ``--spacing %s``
transform_matrix: (a list of items which are a float)
    12 parameters of the transform matrix by rows ( --last 3 being
    translation-- )
    argument: ``--transform_matrix %s``
rotation_point: (a list of items which are any value)
    Rotation Point in case of rotation around a point (otherwise
    useless)
    argument: ``--rotation_point %s``
transformationFile: (an existing file name)
    argument: ``--transformationFile %s``
transform: ('rt' or 'a')
    Transform algorithm, rt = Rigid Transform, a = Affine Transform
    argument: ``--transform %s``
spaceChange: (a boolean)
    Space Orientation between transform and image is different (RAS/LPS)

```

(continues on next page)

(continued from previous page)

```

        (warning: if the transform is a Transform Node in Slicer3, do not
        select)
        argument: ``--spaceChange ``
origin: (a list of items which are any value)
        Origin of the output Image
        argument: ``--origin %s``
spline_order: (an integer (int or long))
        Spline Order
        argument: ``--spline_order %d``
size: (a list of items which are a float)
        Size along each dimension (0 means use input size)
        argument: ``--size %s``
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)
        argument: ``--direction_matrix %s``
default_pixel_value: (a float)
        Default pixel value for samples falling outside of the input region
        argument: ``--default_pixel_value %f``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
transform_order: ('input-to-output' or 'output-to-input')
        Select in what order the transforms are read
        argument: ``--transform_order %s``
outputVolume: (a boolean or a file name)
        Resampled Volume
        argument: ``%s``, position: -1

```

Outputs:

```

outputVolume: (an existing file name)
        Resampled Volume

```

76.13 interfaces.slicer.filtering.thresholdscalervolume

76.13.1 ThresholdScalarVolume

[Link to code](#)

Wraps the executable 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:

```
[Optional]
upper: (an integer (int or long))
    Upper threshold value
    argument: ``--upper %d``
InputVolume: (an existing file name)
    Input volume
    argument: ``%s``, position: -2
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
threshold: (an integer (int or long))
    Threshold value
    argument: ``--threshold %d``
lower: (an integer (int or long))
    Lower threshold value
    argument: ``--lower %d``
outsidevalue: (an integer (int or long))
    Set the voxels to this value if they fall outside the threshold
    range
    argument: ``--outsidevalue %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.
    argument: ``--thresholdtype %s``
environ: (a dictionary with keys which are a 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
OutputVolume: (a boolean or a file name)
    Thresholded input volume
    argument: ``%s``, position: -1
```

Outputs:

```
OutputVolume: (an existing file name)
    Thresholded input volume
```

76.14 interfaces.slicer.filtering.votingbinaryholefillingimagefilter

76.14.1 VotingBinaryHoleFillingImageFilter

Link to code

Wraps the executable 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:

```
[Optional]
foreground: (an integer (int or long))
    The value associated with the foreground (object)
    argument: ``--foreground %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
radius: (a list of items which are an integer (int or long))
    The radius of a hole to be filled
    argument: ``--radius %s``
inputVolume: (an existing file name)
    Input volume to be filtered
    argument: ``%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.
    argument: ``--majorityThreshold %d``
background: (an integer (int or long))
    The value associated with the background (not object)
    argument: ``--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
outputVolume: (a boolean or a file name)
    Output filtered
    argument: ``%s``, position: -1
```

Outputs:

```
outputVolume: (an existing file name)
    Output filtered
```

76.15 interfaces.slicer.legacy.converters

76.15.1 BSplineToDeformationField

[Link to code](#)

Wraps the executable 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:

```
[Optional]
tfm: (an existing file name)
    argument: ``--tfm %s``
```

(continues on next page)

(continued from previous page)

```

refImage: (an existing file name)
    argument: ``--refImage %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
defImage: (a boolean or a file name)
    argument: ``--defImage %s``

```

Outputs:

```
defImage: (an existing file name)
```

76.16 interfaces.slicer.legacy.diffusion.denoising

76.16.1 DWIUnbiasedNonLocalMeansFilter

[Link to code](#)

Wraps the executable 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:

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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
    argument: ``--hp %f``
ng: (an integer (int or long))
    The number of the closest gradients that are used to jointly filter

```

(continues on next page)

(continued from previous page)

```

    a given gradient direction (a maximum of 5 is allowed).
    argument: ``--ng %d``
rc: (a list of items which are an integer (int or long))
    Similarity between blocks is measured using windows of this size.
    argument: ``--rc %s``
inputVolume: (an existing file name)
    Input DWI volume.
    argument: ``%s``, position: -2
outputVolume: (a boolean or a file name)
    Output DWI volume.
    argument: ``%s``, position: -1
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).
    argument: ``--rs %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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.
    argument: ``--re %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Output DWI volume.

```

76.17 interfaces.slicer.legacy.filtering

76.17.1 OtsuThresholdImageFilter

[Link to code](#)

Wraps the executable 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/OtsuThresholdImageFilter>

contributor: Bill Lorensen (GE)

acknowledgements: This command module was derived from Insight/Examples (copyright) Insight Software Consortium

Inputs:

```
[Optional]
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.
    argument: ``--numberOfBins %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
insideValue: (an integer (int or long))
    The value assigned to pixels that are inside the computed threshold
    argument: ``--insideValue %d``
inputVolume: (an existing file name)
    Input volume to be filtered
    argument: ``%s``, position: -2
outsideValue: (an integer (int or long))
    The value assigned to pixels that are outside the computed threshold
    argument: ``--outsideValue %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
outputVolume: (a boolean or a file name)
    Output filtered
    argument: ``%s``, position: -1
```

Outputs:

```
outputVolume: (an existing file name)
    Output filtered
```

76.17.2 ResampleScalarVolume

Link to code

Wraps the executable 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:

```
[Optional]
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;
argument: ``--interpolation %s``
OutputVolume: (a boolean or a file name)
Resampled Volume
argument: ``%s``, position: -1
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
spacing: (a list of items which are a float)
Spacing along each dimension (0 means use input spacing)
argument: ``--spacing %s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
Environment variables
InputVolume: (an existing file name)
Input volume to be resampled
argument: ``%s``, position: -2
```

Outputs:

```
OutputVolume: (an existing file name)
Resampled Volume
```

76.18 interfaces.slicer.legacy.registration

76.18.1 AffineRegistration

[Link to code](#)

Wraps the executable 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:


```

[Optional]
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.
    argument: ``--initialtransform %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).
    argument: ``--resampledmovingfilename %s``
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.
    argument: ``--fixedsmoothingfactor %d``
translationscale: (a float)
    Relative scale of translations to rotations, i.e. a value of 100
    means 10mm = 1 degree. (Actual scale used is
     $1/(\text{TranslationScale}^2)$ ). This parameter is used to 'weight' or
    'standardized' the transform parameters and their effect on the
    registration objective function.
    argument: ``--translationscale %f``
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.
    argument: ``--spatialsamples %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.
    argument: ``--movingsmoothingfactor %d``
MovingImageFileName: (an existing file name)
    Moving image
    argument: ``%s``, position: -1
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.
    argument: ``--histogrambins %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
FixedImageFileName: (an existing file name)
    Fixed image to which to register
    argument: ``%s``, position: -2
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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

```

(continues on next page)

(continued from previous page)

```

both).
argument: ``--outputtransform %s``
iterations: (an integer (int or long))
Number of iterations
argument: ``--iterations %d``

```

Outputs:

```

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).
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).

```

76.18.2 BSplineDeformableRegistration**Link to code**

Wraps the executable 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:

```

[Optional]
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.
argument: ``--initialtransform %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).
argument: ``--resampledmovingfilename %s``
maximumDeformation: (a float)
If Constrain Deformation is checked, limit the deformation to this
amount.
argument: ``--maximumDeformation %f``
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.
argument: ``--spatialsamples %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

```

(continues on next page)

(continued from previous page)

```

        impulses and will inhibit reliable registration estimation.
        argument: ``--histogrambins %d``
gridSize: (an integer (int or long))
        Number of grid points on interior of the fixed image. Larger grid
        sizes allow for finer registrations.
        argument: ``--gridSize %d``
MovingImageFileName: (an existing file name)
        Moving image
        argument: ``%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
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).
        argument: ``--outputtransform %s``
FixedImageFileName: (an existing file name)
        Fixed image to which to register
        argument: ``%s``, position: -2
default: (an integer (int or long))
        Default pixel value used if resampling a pixel outside of the
        volume.
        argument: ``--default %d``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
constrain: (a boolean)
        Constrain the deformation to the amount specified in Maximum
        Deformation
        argument: ``--constrain ``
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.
        argument: ``--outputwarp %s``
iterations: (an integer (int or long))
        Number of iterations
        argument: ``--iterations %d``

```

Outputs:

```

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.
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).
resampledmovingfilename: (an existing file name)
        Resampled moving image to fixed image coordinate frame. Optional
        (specify an output transform or an output volume or both).

```

76.18.3 ExpertAutomatedRegistration

[Link to code](#)

Wraps the executable 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/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:

```
[Optional]
sampleFromOverlap: (a boolean)
    Limit metric evaluation to the fixed image region overlapped by the
    moving image
    argument: ``--sampleFromOverlap ``
fixedImageMask: (an existing file name)
    Image which defines a mask for the fixed image
    argument: ``--fixedImageMask %s``
affineMaxIterations: (an integer (int or long))
    Maximum number of affine optimization iterations
    argument: ``--affineMaxIterations %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
fixedImage: (an existing file name)
    Image which defines the space into which the moving image is
    registered
    argument: ``%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', nipyte default value: {})
    Environment variables
expectedRotation: (a float)
    Expected misalignment after initialization
    argument: ``--expectedRotation %f``
loadTransform: (an existing file name)
    Load a transform that is immediately applied to the moving image
    argument: ``--loadTransform %s``
saveTransform: (a boolean or a file name)
    Save the transform that results from registration
    argument: ``--saveTransform %s``
registration: ('None' or 'Initial' or 'Rigid' or 'Affine' or
    'BSpline' or 'PipelineRigid' or 'PipelineAffine' or
    'PipelineBSpline')
    Method for the registration process
    argument: ``--registration %s``
minimizeMemory: (a boolean)
    Reduce the amount of memory required at the cost of increased
    computation time
    argument: ``--minimizeMemory ``
rigidSamplingRatio: (a float)
    Portion of the image to use in computing the metric during rigid
```

(continues on next page)

(continued from previous page)

```

    registration
    argument: ``--rigidSamplingRatio %f``
expectedScale: (a float)
    Expected misalignment after initialization
    argument: ``--expectedScale %f``
interpolation: ('NearestNeighbor' or 'Linear' or 'BSpline')
    Method for interpolation within the optimization process
    argument: ``--interpolation %s``
initialization: ('None' or 'Landmarks' or 'ImageCenters' or
    'CentersOfMass' or 'SecondMoments')
    Method to prime the registration process
    argument: ``--initialization %s``
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
    argument: ``--movingLandmarks %s...``
bsplineSamplingRatio: (a float)
    Portion of the image to use in computing the metric during BSpline
    registration
    argument: ``--bsplineSamplingRatio %f``
controlPointSpacing: (an integer (int or long))
    Number of pixels between control points
    argument: ``--controlPointSpacing %d``
rigidMaxIterations: (an integer (int or long))
    Maximum number of rigid optimization iterations
    argument: ``--rigidMaxIterations %d``
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
    argument: ``--fixedLandmarks %s...``
movingImage: (an existing file name)
    The transform goes from the fixed image's space into the moving
    image's space
    argument: ``%s``, position: -1
expectedSkew: (a float)
    Expected misalignment after initialization
    argument: ``--expectedSkew %f``
affineSamplingRatio: (a float)
    Portion of the image to use in computing the metric during affine
    registration
    argument: ``--affineSamplingRatio %f``
metric: ('MattesMI' or 'NormCorr' or 'MeanSqr')
    Method to quantify image match
    argument: ``--metric %s``
numberOfThreads: (an integer (int or long))
    Number of CPU threads to use
    argument: ``--numberOfThreads %d``
verbosityLevel: ('Silent' or 'Standard' or 'Verbose')
    Level of detail of reporting progress
    argument: ``--verbosityLevel %s``
randomNumberSeed: (an integer (int or long))
    Seed to generate a consistent random number sequence
    argument: ``--randomNumberSeed %d``
resampledImage: (a boolean or a file name)
    Registration results
    argument: ``--resampledImage %s``
bsplineMaxIterations: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

Maximum number of bspline optimization iterations
argument: ``--bsplineMaxIterations %d``
expectedOffset: (a float)
Expected misalignment after initialization
argument: ``--expectedOffset %f``

```

Outputs:

```

resampledImage: (an existing file name)
Registration results
saveTransform: (an existing file name)
Save the transform that results from registration

```

76.18.4 LinearRegistration[Link to code](#)

Wraps the executable 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:

```

[Optional]
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.
argument: ``--initialtransform %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.
argument: ``--learningrate %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).
argument: ``--resampledmovingfilename %s``
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.
argument: ``--fixedsmoothingfactor %d``
translationscale: (a float)

```

(continues on next page)

(continued from previous page)

```

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.
argument: ``--translationscale %f``
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.
argument: ``--spatialsamples %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.
argument: ``--movingsmoothingfactor %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.
argument: ``--histogrambins %d``
MovingImageFileName: (an existing file name)
Moving image
argument: ``%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
FixedImageFileName: (an existing file name)
Fixed image to which to register
argument: ``%s``, position: -2
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
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).
argument: ``--outputtransform %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.
argument: ``--iterations %s``

```

Outputs:

```

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).
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).

```

76.18.5 MultiResolutionAffineRegistration

[Link to code](#)

Wraps the executable 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:

```
[Optional]
fixedImageMask: (an existing file name)
    Label image which defines a mask of interest for the fixed image
    argument: ``--fixedImageMask %s``
fixedImageROI: (a list of items which are any value)
    Label image which defines a ROI of interest for the fixed image
    argument: ``--fixedImageROI %s``
fixedImage: (an existing file name)
    Image which defines the space into which the moving image is
    registered
    argument: ``%s``, position: -2
numLineIterations: (an integer (int or long))
    Number of iterations to run at each resolution level.
    argument: ``--numLineIterations %d``
metricTolerance: (a float)
    argument: ``--metricTolerance %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
saveTransform: (a boolean or a file name)
    Save the output transform from the registration
    argument: ``--saveTransform %s``
stepSize: (a float)
    The maximum step size of the optimizer in voxels
    argument: ``--stepSize %f``
movingImage: (an existing file name)
    The transform goes from the fixed image's space into the moving
    image's space
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
stepTolerance: (a float)
    The maximum step size of the optimizer in voxels
    argument: ``--stepTolerance %f``
numIterations: (an integer (int or long))
    Number of iterations to run at each resolution level.
    argument: ``--numIterations %d``
resampledImage: (a boolean or a file name)
    Registration results
    argument: ``--resampledImage %s``
```

Outputs:


```
resampledImage: (an existing file name)
    Registration results
saveTransform: (an existing file name)
    Save the output transform from the registration
```

76.18.6 RigidRegistration

[Link to code](#)

Wraps the executable 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:

```
[Optional]
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.
    argument: ``--initialtransform %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.
    argument: ``--learningrate %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).
    argument: ``--resampledmovingfilename %s``
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.
```

(continues on next page)

(continued from previous page)

```

        argument: ``--fixedsmoothingfactor %d``
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.
    argument: ``--translationscale %f``
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.
    argument: ``--spatialsamples %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.
    argument: ``--movingsmoothingfactor %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.
    argument: ``--histogrambins %d``
MovingImageFileName: (an existing file name)
    Moving image
    argument: ``%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
FixedImageFileName: (an existing file name)
    Fixed image to which to register
    argument: ``%s``, position: -2
testingmode: (a boolean)
    Enable testing mode. Input transform will be used to construct
    floating image. The floating image will be ignored if passed.
    argument: ``--testingmode``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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).
    argument: ``--outputtransform %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.
    argument: ``--iterations %s``

```

Outputs:

```

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).

```

(continues on next page)

(continued from previous page)

```

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).

```

76.19 interfaces.slicer.legacy.segmentation

76.19.1 OtsuThresholdSegmentation

[Link to code](#)

Wraps the executable 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:

```

[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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.
    argument: ``--numberOfBins %d``
brightObjects: (a boolean)
    Segmenting bright objects on a dark background or dark objects on a
    bright background.
    argument: ``--brightObjects ``
inputVolume: (an existing file name)
    Input volume to be segmented
    argument: ``%s``, position: -2
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.
    argument: ``--faceConnected ``
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.

```

(continues on next page)

(continued from previous page)

```

        argument: ``--minimumObjectSize %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
outputVolume: (a boolean or a file name)
        Output filtered
        argument: ``%s``, position: -1

```

Outputs:

```

outputVolume: (an existing file name)
        Output filtered

```

76.20 interfaces.slicer.quantification.changequantification

76.20.1 IntensityDifferenceMetric

[Link to code](#)

Wraps the executable 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:

```

[Optional]
followupVolume: (an existing file name)
        Followup volume to be compare to the baseline
        argument: ``%s``, position: -2
baselineVolume: (an existing file name)
        Baseline volume to be compared to
        argument: ``%s``, position: -4
baselineSegmentationVolume: (an existing file name)
        Label volume that contains segmentation of the structure of interest
        in the baseline volume.
        argument: ``%s``, position: -3
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
reportFileName: (a boolean or a file name)
        Report file name
        argument: ``--reportFileName %s``
outputVolume: (a boolean or a file name)
        Output volume to keep the results of change quantification.
        argument: ``%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
changingBandSize: (an integer (int or long))
        How far (in mm) from the boundary of the segmentation should the

```

(continues on next page)

(continued from previous page)

```

intensity changes be considered.
argument: ``--changingBandSize %d``
sensitivityThreshold: (a float)
This parameter should be between 0 and 1, and defines how sensitive
the metric should be to the intensity changes.
argument: ``--sensitivityThreshold %f``

```

Outputs:

```

reportFileName: (an existing file name)
    Report file name
outputVolume: (an existing file name)
    Output volume to keep the results of change quantification.

```

76.21 interfaces.slicer.quantification.petstandarduptakevaluecomputation

76.21.1 PETStandardUptakeValueComputation

Link to code

Wraps the executable 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/](http://www.slicer.org/slicerWiki/index.php/Documentation/4.1/Modules/ComputeSUVBodyWeight)

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:

```

[Optional]
labelMap: (an existing file name)
    Input label volume containing the volumes of interest
    argument: ``--labelMap %s``
SUVMax: (a unicode string)
    SUV max for each label
    argument: ``--SUVMax %s``
csvFile: (a boolean or a file name)
    A file holding the output SUV values in comma separated lines, one
    per label. Optional.
    argument: ``--csvFile %s``
color: (an existing file name)
    Color table to map labels to colors and names
    argument: ``--color %s``
petDICOMPath: (an existing directory name)
    Input path to a directory containing a PET volume containing DICOM
    header information for SUV computation
    argument: ``--petDICOMPath %s``

```

(continues on next page)

(continued from previous page)

```

environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
OutputLabel: (a unicode string)
         List of labels for which SUV values were computed
         argument: ``--OutputLabel %s``
OutputLabelValue: (a unicode string)
         List of label values for which SUV values were computed
         argument: ``--OutputLabelValue %s``
args: (a unicode string)
         Additional parameters to the command
         argument: ``%s``
SUVMin: (a unicode string)
         SUV minimum for each label
         argument: ``--SUVMin %s``
petVolume: (an existing file name)
         Input PET volume for SUVbw computation (must be the same volume as
         pointed to by the DICOM path!).
         argument: ``--petVolume %s``
SUVMean: (a unicode string)
         SUV mean for each label
         argument: ``--SUVMean %s``

```

Outputs:

```

csvFile: (an existing file name)
         A file holding the output SUV values in comma separated lines, one
         per label. Optional.

```

76.22 interfaces.slicer.registration.brainsfit

76.22.1 BRAINSFit

[Link to code](#)

Wraps the executable 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:

```

[Optional]
failureExitCode: (an integer (int or long))
                 If the fit fails, exit with this status code. (It can be used to

```

(continues on next page)

(continued from previous page)

```

    force a successful exit status of (0) if the registration fails due
    to reaching the maximum number of iterations.
    argument: ``--failureExitCode %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
    argument: ``--numberOfIterations %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
    argument: ``--linearTransform %s``
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 modalities.
    argument: ``--histogramMatch ``
outputMovingVolumeROI: (a boolean or a file name)
    The ROI automatically found in moving image, ONLY FOR ROIAUTO mode.
    argument: ``--outputMovingVolumeROI %s``
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.
    argument: ``--scaleOutputValues ``
useRigid: (a boolean)
    Perform a rigid registration as part of the sequential registration
    steps. This family of options supercedes the use of transformType if
    any of them are set.
    argument: ``--useRigid ``
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
    argument: ``--skewScale %f``
debugLevel: (an integer (int or long))
    Display debug messages, and produce debug intermediate results.
    0=OFF, 1=Minimal, 10=Maximum debugging.
    argument: ``--debugLevel %d``
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.
    argument: ``--splineGridSize %s``
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
    argument: ``--bsplineTransform %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

```

(continues on next page)

(continued from previous page)

```

    linearTransform is set.
    argument: ``--strippedOutputTransform %s``
ROIAutoClosingSize: (a float)
    This flag is only relavent 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.
    argument: ``--ROIAutoClosingSize %f``
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.
    argument: ``--interpolationMode %s``
medianFilterSize: (a list of items which are an integer (int or
    long))
    The radius for the optional MedianImageFilter preprocessing in all 3
    directions.
    argument: ``--medianFilterSize %s``
maximumStepLength: (a float)
    Internal debugging parameter, and should probably never be used from
    the command line. This will be removed in the future.
    argument: ``--maximumStepLength %f``
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.
    argument: ``--translationScale %f``
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.
    argument: ``--numberOfSamples %d``
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.
    Specifiying 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.
    argument: ``--transformType %s``
numberOfMatchPoints: (an integer (int or long))
    the number of match points
    argument: ``--numberOfMatchPoints %d``
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
    argument: ``--reproportionScale %f``
NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_00: (a boolean)
    DO NOT USE THIS FLAG
    argument: ``--NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_00 ``
initialTransform: (an existing file name)
    Filename of transform used to initialize the registration. This CAN

```

(continues on next page)

(continued from previous page)

```

NOT be used with either CenterOfHeadLAlign, MomentsAlign,
GeometryAlign, or initialTransform file.
argument: ``--initialTransform %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use. (default is
    auto-detected)
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
useCachingOfBSplineWeightsMode: ('ON' or 'OFF')
    This is a 5x speed advantage at the expense of requiring much more
    memory. Only relevant when transformType is BSpline.
    argument: ``--useCachingOfBSplineWeightsMode %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.,
    argument: ``--costFunctionConvergenceFactor %f``
NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_02: (a boolean)
    DO NOT USE THIS FLAG
    argument: ``--NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_02 ``
movingVolumeTimeIndex: (an integer (int or long))
    The index in the time series for the 3D moving image to fit, if
    4-dimensional.
    argument: ``--movingVolumeTimeIndex %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.
    argument: ``--gui ``
outputVolume: (a boolean or a file name)
    (optional) Output image for registration. NOTE: You must select
    either the outputTransform or the outputVolume option.
    argument: ``--outputVolume %s``
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels
    argument: ``--numberOfHistogramBins %d``
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.
    argument: ``--useScaleVersor3D ``
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.
    argument: ``--useBSpline ``
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)
    argument: ``--costMetric %s``
fixedBinaryVolume: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

Fixed Image binary mask volume, ONLY FOR MANUAL ROI mode.
argument: ``--fixedBinaryVolume %s``
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.
argument: ``--useScaleSkewVersor3D``
outputFixedVolumeROI: (a boolean or a file name)
The ROI automatically found in fixed image, ONLY FOR ROIAUTO mode.
argument: ``--outputFixedVolumeROI %s``
relaxationFactor: (a float)
Internal debugging parameter, and should probably never be used from
the command line. This will be removed in the future.
argument: ``--relaxationFactor %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
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.
argument: ``--outputTransform %s``
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.
argument: ``--writeOutputTransformInFloat``
movingVolume: (an existing file name)
The moving image for registration by mutual information
optimization.
argument: ``--movingVolume %s``
maskInferiorCutOffFromCenter: (a float)
For use with --useCenterOfHeadAlign (and --maskProcessingMode
ROIAUTO): the cut-off below the image centers, in millimeters,
argument: ``--maskInferiorCutOffFromCenter %f``
NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_01: (a boolean)
DO NOT USE THIS FLAG
argument: ``--NEVER_USE_THIS_FLAG_IT_IS_OUTDATED_01``
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.
argument: ``--minimumStepLength %s``
fixedVolumeTimeIndex: (an integer (int or long))
The index in the time series for the 3D fixed image to fit, if
4-dimensional.
argument: ``--fixedVolumeTimeIndex %d``
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

```

(continues on next page)

(continued from previous page)

```

    reduction in computation time in the BSpline optimizer.,
    argument: ``--maxBSplineDisplacement %f``
writeTransformOnFailure: (a boolean)
    Flag to save the final transform even if the numberOfIterations are
    reached without convergence. (Intended for use when
    --failureExitCode 0 )
    argument: ``--writeTransformOnFailure ``
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.
    argument: ``--useExplicitPDFDerivativesMode %s``
fixedVolume: (an existing file name)
    The fixed image for registration by mutual information optimization.
    argument: ``--fixedVolume %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.
    argument: ``--permitParameterVariation %s``
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.
    argument: ``--ROIAutoDilateSize %f``
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.,
    argument: ``--projectedGradientTolerance %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.
    argument: ``--maskProcessingMode %s``
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    argument: ``--promptUser ``
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

```

(continues on next page)

(continued from previous page)

```

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.
argument: ``--initializeTransformMode %s``
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.
argument: ``--removeIntensityOutliers %f``
backgroundFillValue: (a float)
Background fill value for output image.
argument: ``--backgroundFillValue %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.
argument: ``--useAffine``
forceMINumberOfThreads: (an integer (int or long))
Force the the maximum number of threads to use for non thread safe
MI metric. CAUTION: Inconsistent results may arise!
argument: ``--forceMINumberOfThreads %d``
movingBinaryVolume: (an existing file name)
Moving Image binary mask volume, ONLY FOR MANUAL ROI mode.
argument: ``--movingBinaryVolume %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.
argument: ``--outputVolumePixelType %s``

```

Outputs:

```

outputMovingVolumeROI: (an existing file name)
The ROI automatically found in moving image, ONLY FOR ROIAUTO mode.
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
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.
outputFixedVolumeROI: (an existing file name)
The ROI automatically found in fixed image, ONLY FOR ROIAUTO mode.
outputVolume: (an existing file name)
(optional) Output image for registration. NOTE: You must select

```

(continues on next page)

(continued from previous page)

```

    either the outputTransform or the outputVolume option.
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.

```

76.23 interfaces.slicer.registration.brainsresample

76.23.1 BRAINSResample

[Link to code](#)

Wraps the executable 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:

```

[Optional]
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
    argument: ``--interpolationMode %s``
defaultValue: (a float)
    Default voxel value
    argument: ``--defaultValue %f``
inputVolume: (an existing file name)
    Image To Warp
    argument: ``--inputVolume %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.
    argument: ``--referenceVolume %s``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
inverseTransform: (a boolean)
    True/False is to compute inverse of given transformation. Default is
    false
    argument: ``--inverseTransform ``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.

```

(continues on next page)

(continued from previous page)

```

        argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--pixelType %s``
deformationVolume: (an existing file name)
    Displacement Field to be used to warp the image
    argument: ``--deformationVolume %s``
warpTransform: (an existing file name)
    Filename for the BRAINSFit transform used in place of the
    deformation field
    argument: ``--warpTransform %s``
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
    argument: ``--gridSpacing %s``
outputVolume: (a boolean or a file name)
    Resulting deformed image
    argument: ``--outputVolume %s``

```

Outputs:

```

outputVolume: (an existing file name)
    Resulting deformed image

```

76.24 interfaces.slicer.registration.specialized

76.24.1 ACPCTransform

[Link to code](#)

Wraps the executable 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 Volumeto 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:

```
[Optional]
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).
         argument: ``--midline %s...``
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.
      argument: ``--acpc %s...``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
debugSwitch: (a boolean)
      Click if wish to see debugging output
      argument: ``--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', nipype default value: {})
         Environment variables
outputTransform: (a boolean or a file name)
         A transform filled in from the ACPC and Midline registration
         calculation
         argument: ``--outputTransform %s``
```

Outputs:

```
outputTransform: (an existing file name)
                 A transform filled in from the ACPC and Midline registration
                 calculation
```

76.24.2 BRAINSDemonWarp

Link to code

Wraps the executable 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:

```
[Optional]
numberOfPyramidLevels: (an integer (int or long))
                       Number of image pyramid levels to use in the multi-resolution
                       registration.
```

(continues on next page)

(continued from previous page)

```

        argument: ``--numberOfPyramidLevels %d``
max_step_length: (a float)
    Maximum length of an update vector (0: no restriction)
    argument: ``--max_step_length %f``
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels
    argument: ``--numberOfHistogramBins %d``
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)
    argument: ``--gradient_type %s``
outputDisplacementFieldVolume: (a boolean or a file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
    argument: ``--outputDisplacementFieldVolume %s``
seedForBOBF: (a list of items which are an integer (int or long))
    coordinates in all 3 directions for Seed when performing BOBF
    argument: ``--seedForBOBF %s``
outputDisplacementFieldPrefix: (a unicode string)
    Displacement field filename prefix for writing separate x, y, and z
    component images
    argument: ``--outputDisplacementFieldPrefix %s``
fixedBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Fixed image.
    argument: ``--fixedBinaryVolume %s``
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.
    argument: ``--histogramMatch ``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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)
    argument: ``--minimumMovingPyramid %s``
numberOfMatchPoints: (an integer (int or long))
    The number of match points for histogramMatch
    argument: ``--numberOfMatchPoints %d``
numberOfBCHApproximationTerms: (an integer (int or long))
    Number of terms in the BCH expansion
    argument: ``--numberOfBCHApproximationTerms %d``
use_vanilla_dem: (a boolean)
    Run vanilla demons algorithm
    argument: ``--use_vanilla_dem ``
checkerboardPatternSubdivisions: (a list of items which are an
    integer (int or long))
    Number of Checkerboard subdivisions in all 3 directions
    argument: ``--checkerboardPatternSubdivisions %s``
outputDebug: (a boolean)
    Flag to write debugging images after each step.
    argument: ``--outputDebug ``
outputNormalized: (a boolean)

```

(continues on next page)

(continued from previous page)

```

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.
argument: ``--outputNormalized ``
neighborhoodForBOBF: (a list of items which are an integer (int or
long))
neighborhood in all 3 directions to be included when performing BOBF
argument: ``--neighborhoodForBOBF %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
argument: ``--interpolationMode %s``
lowerThresholdForBOBF: (an integer (int or long))
Lower threshold for performing BOBF
argument: ``--lowerThresholdForBOBF %d``
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.
argument: ``--medianFilterSize %s``
upperThresholdForBOBF: (an integer (int or long))
Upper threshold for performing BOBF
argument: ``--upperThresholdForBOBF %d``
fixedVolume: (an existing file name)
Required: input fixed (target) image
argument: ``--fixedVolume %s``
outputVolume: (a boolean or a file name)
Required: output resampled moving image (will have the same physical
space as the fixedVolume).
argument: ``--outputVolume %s``
inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
Input volumes will be typecast to this format:
float|short|ushort|int|uchar
argument: ``--inputPixelType %s``
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.
argument: ``--maskProcessingMode %s``
arrayOfPyramidLevelIterations: (a list of items which are an integer
(int or long))
The number of iterations for each pyramid level
argument: ``--arrayOfPyramidLevelIterations %s``
registrationFilterType: ('Demons' or 'FastSymmetricForces' or
'Diffeomorphic')
Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic
argument: ``--registrationFilterType %s``
outputCheckerboardVolume: (a boolean or a file name)
Genete a checkerboard image volume between the fixedVolume and the
deformed movingVolume.

```

(continues on next page)

(continued from previous page)

```

        argument: ``--outputCheckerboardVolume %s``
initializeWithDisplacementField: (an existing file name)
    Initial deformation field vector image file name
    argument: ``--initializeWithDisplacementField %s``
gui: (a boolean)
    Display intermediate image volumes for debugging
    argument: ``--gui ``
initializeWithTransform: (an existing file name)
    Initial Transform filename
    argument: ``--initializeWithTransform %s``
backgroundFillValue: (an integer (int or long))
    Replacement value to overwrite background when performing BOBF
    argument: ``--backgroundFillValue %d``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
movingVolume: (an existing file name)
    Required: input moving image
    argument: ``--movingVolume %s``
upFieldSmoothing: (a float)
    Smoothing sigma for the update field at each iteration
    argument: ``--upFieldSmoothing %f``
outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    outputVolume will be typecast to this format:
    float|short|ushort|int|uchar
    argument: ``--outputPixelType %s``
movingBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Moving image.
    argument: ``--movingBinaryVolume %s``
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    argument: ``--promptUser ``
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)
    argument: ``--minimumFixedPyramid %s``
smoothDisplacementFieldSigma: (a float)
    A gaussian smoothing value to be applied to the deformation feild at
    each iteration.
    argument: ``--smoothDisplacementFieldSigma %f``

```

Outputs:

```

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).
outputCheckerboardVolume: (an existing file name)
    Genete a checkerboard image volume between the fixedVolume and the

```

(continues on next page)

(continued from previous page)

deformed movingVolume.

76.24.3 FiducialRegistration

[Link to code](#)

Wraps the executable 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:

```
[Optional]
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
                  argument: ``--movingLandmarks %s...``
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
outputMessage: (a unicode string)
               Provides more information on the output
               argument: ``--outputMessage %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
                 argument: ``--fixedLandmarks %s...``
transformType: ('Translation' or 'Rigid' or 'Similarity')
               Type of transform to produce
               argument: ``--transformType %s``
rms: (a float)
     Display RMS Error.
     argument: ``--rms %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', nipype default value: {})
          Environment variables
saveTransform: (a boolean or a file name)
               Save the transform that results from registration
               argument: ``--saveTransform %s``
```

Outputs:

```
saveTransform: (an existing file name)
               Save the transform that results from registration
```

76.24.4 VBRAINSDemonWarp

[Link to code](#)

Wraps the executable 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: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:

```
[Optional]
numberOfPyramidLevels: (an integer (int or long))
    Number of image pyramid levels to use in the multi-resolution
    registration.
    argument: ``--numberOfPyramidLevels %d``
max_step_length: (a float)
    Maximum length of an update vector (0: no restriction)
    argument: ``--max_step_length %f``
numberOfHistogramBins: (an integer (int or long))
    The number of histogram levels
    argument: ``--numberOfHistogramBins %d``
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)
    argument: ``--gradient_type %s``
outputDisplacementFieldVolume: (a boolean or a file name)
    Output deformation field vector image (will have the same physical
    space as the fixedVolume).
    argument: ``--outputDisplacementFieldVolume %s``
seedForBOBF: (a list of items which are an integer (int or long))
    coordinates in all 3 directions for Seed when performing BOBF
    argument: ``--seedForBOBF %s``
lowerThresholdForBOBF: (an integer (int or long))
    Lower threshold for performing BOBF
    argument: ``--lowerThresholdForBOBF %d``
fixedBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Fixed image.
    argument: ``--fixedBinaryVolume %s``
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.
    argument: ``--histogramMatch``
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
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)
    argument: ``--minimumMovingPyramid %s``
```

(continues on next page)

(continued from previous page)

```

numberOfMatchPoints: (an integer (int or long))
    The number of match points for histogramMatch
    argument: ``--numberOfMatchPoints %d``
numberOfBCHApproximationTerms: (an integer (int or long))
    Number of terms in the BCH expansion
    argument: ``--numberOfBCHApproximationTerms %d``
registrationFilterType: ('Demons' or 'FastSymmetricForces' or
    'Diffeomorphic' or 'LogDemons' or 'SymmetricLogDemons')
    Registration Filter Type: Demons|FastSymmetricForces|Diffeomorphic|L
    ogDemons|SymmetricLogDemons
    argument: ``--registrationFilterType %s``
use_vanilla_dem: (a boolean)
    Run vanilla demons algorithm
    argument: ``--use_vanilla_dem``
checkerboardPatternSubdivisions: (a list of items which are an
    integer (int or long))
    Number of Checkerboard subdivisions in all 3 directions
    argument: ``--checkerboardPatternSubdivisions %s``
outputDebug: (a boolean)
    Flag to write debugging images after each step.
    argument: ``--outputDebug``
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.
    argument: ``--outputNormalized``
neighborhoodForBOBF: (a list of items which are an integer (int or
    long))
    neighborhood in all 3 directions to be included when performing BOBF
    argument: ``--neighborhoodForBOBF %s``
weightFactors: (a list of items which are a float)
    Weight fatctors for each input images
    argument: ``--weightFactors %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
    argument: ``--interpolationMode %s``
initializeWithDisplacementField: (an existing file name)
    Initial deformation field vector image file name
    argument: ``--initializeWithDisplacementField %s``
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.
    argument: ``--medianFilterSize %s``
upperThresholdForBOBF: (an integer (int or long))
    Upper threshold for performing BOBF
    argument: ``--upperThresholdForBOBF %d``
fixedVolume: (a list of items which are an existing file name)
    Required: input fixed (target) image
    argument: ``--fixedVolume %s...``
outputDisplacementFieldPrefix: (a unicode string)
    Displacement field filename prefix for writing separate x, y, and z
    component images

```

(continues on next page)

(continued from previous page)

```

        argument: ``--outputDisplacementFieldPrefix %s``
inputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    Input volumes will be typecast to this format:
    float|short|ushort|int|uchar
    argument: ``--inputPixelType %s``
arrayOfPyramidLevelIterations: (a list of items which are an integer
    (int or long))
    The number of iterations for each pyramid level
    argument: ``--arrayOfPyramidLevelIterations %s``
makeBOBF: (a boolean)
    Flag to make Brain-Only Background-Filled versions of the input and
    target volumes.
    argument: ``--makeBOBF ``
outputCheckerboardVolume: (a boolean or a file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.
    argument: ``--outputCheckerboardVolume %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)
    argument: ``--minimumFixedPyramid %s``
gui: (a boolean)
    Display intermediate image volumes for debugging
    argument: ``--gui ``
initializeWithTransform: (an existing file name)
    Initial Transform filename
    argument: ``--initializeWithTransform %s``
backgroundFillValue: (an integer (int or long))
    Replacement value to overwrite background when performing BOBF
    argument: ``--backgroundFillValue %d``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
movingVolume: (a list of items which are an existing file name)
    Required: input moving image
    argument: ``--movingVolume %s...``
upFieldSmoothing: (a float)
    Smoothing sigma for the update field at each iteration
    argument: ``--upFieldSmoothing %f``
outputPixelType: ('float' or 'short' or 'ushort' or 'int' or 'uchar')
    outputVolume will be typecast to this format:
    float|short|ushort|int|uchar
    argument: ``--outputPixelType %s``
movingBinaryVolume: (an existing file name)
    Mask filename for desired region of interest in the Moving image.
    argument: ``--movingBinaryVolume %s``
promptUser: (a boolean)
    Prompt the user to hit enter each time an image is sent to the
    DebugImageViewer
    argument: ``--promptUser ``
outputVolume: (a boolean or a file name)
    Required: output resampled moving image (will have the same physical

```

(continues on next page)

(continued from previous page)

```

    space as the fixedVolume).
    argument: ``--outputVolume %s``
smoothDisplacementFieldSigma: (a float)
    A gaussian smoothing value to be applied to the deformation feild at
    each iteration.
    argument: ``--smoothDisplacementFieldSigma %f``

```

Outputs:

```

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).
outputCheckerboardVolume: (an existing file name)
    Genete a checkerboard image volume between the fixedVolume and the
    deformed movingVolume.

```

76.25 interfaces.slicer.segmentation.simpleregiongrowingsegmentation**76.25.1 SimpleRegionGrowingSegmentation****Link to code**

Wraps the executable 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:

```

[Optional]
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
    argument: ``--seed %s...``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
multiplier: (a float)
    Number of standard deviations to include in intensity model
    argument: ``--multiplier %f``
inputVolume: (an existing file name)
    Input volume to be filtered
    argument: ``%s``, position: -2
labelvalue: (an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

    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
    argument: ``--labelvalue %d``
smoothingIterations: (an integer (int or long))
    Number of smoothing iterations
    argument: ``--smoothingIterations %d``
neighborhood: (an integer (int or long))
    The radius of the neighborhood over which to calculate intensity
    model
    argument: ``--neighborhood %d``
outputVolume: (a boolean or a file name)
    Output filtered
    argument: ``%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
iterations: (an integer (int or long))
    Number of iterations of region growing
    argument: ``--iterations %d``
timestep: (a float)
    Timestep for curvature flow
    argument: ``--timestep %f``

```

Outputs:

```

outputVolume: (an existing file name)
    Output filtered

```

76.26 interfaces.slicer.segmentation.specialized

76.26.1 BRAINSROIAuto

Link to code

Wraps the executable 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:

```

[Optional]
outputROIMaskVolume: (a boolean or a file name)
    The ROI automatically found from the input image.
    argument: ``--outputROIMaskVolume %s``

```

(continues on next page)

(continued from previous page)

```

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.
    argument: ``--closingSize %f``
outputClippedVolumeROI: (a boolean or a file name)
    The inputVolume clipped to the region of the brain mask.
    argument: ``--outputClippedVolumeROI %s``
numberOfThreads: (an integer (int or long))
    Explicitly specify the maximum number of threads to use.
    argument: ``--numberOfThreads %d``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
inputVolume: (an existing file name)
    The input image for finding the largest region filled mask.
    argument: ``--inputVolume %s``
otsuPercentileThreshold: (a float)
    Parameter to the Otsu threshold algorithm.
    argument: ``--otsuPercentileThreshold %f``
ROIAutoDilateSize: (a float)
    This flag is only relavent 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.
    argument: ``--ROIAutoDilateSize %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
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.
    argument: ``--outputVolumePixelType %s``
thresholdCorrectionFactor: (a float)
    A factor to scale the Otsu algorithm's result threshold, in case
    clipping mangles the image.
    argument: ``--thresholdCorrectionFactor %f``

```

Outputs:

```

outputROIMaskVolume: (an existing file name)
    The ROI automatically found from the input image.
outputClippedVolumeROI: (an existing file name)
    The inputVolume clipped to the region of the brain mask.

```

76.26.2 EMSegmentCommandLine[Link to code](#)

Wraps the executable 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:

```
[Optional]
disableCompression: (a boolean)
    Don't use compression when writing result image to disk.
    argument: ``--disableCompression ``
parametersMRMLNodeName: (a unicode string)
    The name of the EMSegment parameters node within the active MRML
    scene. Leave blank for default.
    argument: ``--parametersMRMLNodeName %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.
    argument: ``--targetVolumeFileNames %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 !
    argument: ``--atlasVolumeFileNames %s...``
resultMRMLSceneFileName: (a boolean or a file name)
    Write out the MRML scene after command line substitutions have been
    made.
    argument: ``--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.
    argument: ``--resultStandardVolumeFileName %s``
dontWriteResults: (a boolean)
    Used for testing. Don't actually write the resulting labelmap to
    disk.
    argument: ``--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
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
    argument: ``--registrationAffineType %d``
generateEmptyMRMLSceneAndQuit: (a boolean or a file name)
    Used for testing. Only write a scene with default mrml parameters.
    argument: ``--generateEmptyMRMLSceneAndQuit %s``
keepTempFiles: (a boolean)
    If flag is set then at the end of command the temporary files are
    not removed
    argument: ``--keepTempFiles ``
dontUpdateIntermediateData: (an integer (int or long))
    Disable update of intermediate results. -1: Do not overwrite default
    value. 0: Disable. 1: Enable.
    argument: ``--dontUpdateIntermediateData %d``
taskPreProcessingSetting: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

    Specifies the different task parameter. Leave blank for default.
    argument: ``--taskPreProcessingSetting %s``
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.
    argument: ``--disableMultithreading %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
    argument: ``--registrationDeformableType %d``
intermediateResultsDirectory: (an existing directory name)
    Directory where EMSegmenter will write intermediate data (e.g.,
    aligned atlas data).
    argument: ``--intermediateResultsDirectory %s``
loadAtlasNonCentered: (a boolean)
    Read atlas files non-centered.
    argument: ``--loadAtlasNonCentered ``
registrationPackage: (a unicode string)
    specify the registration package for preprocessing (CMTK or BRAINS
    or PLASTIMATCH or DEMONS)
    argument: ``--registrationPackage %s``
mrmlSceneFileName: (an existing file name)
    Active MRML scene that contains EMSegment algorithm parameters.
    argument: ``--mrmlSceneFileName %s``
verbose: (a boolean)
    Enable verbose output.
    argument: ``--verbose ``
loadTargetCentered: (a boolean)
    Read target files centered.
    argument: ``--loadTargetCentered ``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
resultVolumeFileName: (a boolean or a file name)
    The file name that the segmentation result volume will be written
    to.
    argument: ``--resultVolumeFileName %s``

```

Outputs:

```

resultMRMLSceneFileName: (an existing file name)
    Write out the MRML scene after command line substitutions have been
    made.
generateEmptyMRMLSceneAndQuit: (an existing file name)
    Used for testing. Only write a scene with default mrml parameters.
resultVolumeFileName: (an existing file name)
    The file name that the segmentation result volume will be written
    to.

```

76.26.3 RobustStatisticsSegmenter[Link to code](#)

Wraps the executable 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:

```
[Optional]
labelImageFileName: (an existing file name)
    Label image for initialization
    argument: ``%s``, position: -2
curvatureWeight: (a float)
    Given sphere 1.0 score and extreme rough boundary/surface 0 score,
    what is the expected smoothness of the object?
    argument: ``--curvatureWeight %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
segmentedImageFileName: (a boolean or a file name)
    Segmented image
    argument: ``%s``, position: -1
labelValue: (an integer (int or long))
    Label value of the output image
    argument: ``--labelValue %d``
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.
    argument: ``--intensityHomogeneity %f``
originalImageFileName: (an existing file name)
    Original image to be segmented
    argument: ``%s``, position: -3
maxRunningTime: (a float)
    The program will stop if this time is reached.
    argument: ``--maxRunningTime %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', nipype default value: {})
    Environment variables
expectedVolume: (a float)
    The approximate volume of the object, in mL.
    argument: ``--expectedVolume %f``
```

Outputs:

```
segmentedImageFileName: (an existing file name)
    Segmented image
```

76.27 interfaces.slicer.surface

76.27.1 GrayscaleModelMaker

[Link to code](#)

Wraps the executable 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:

```
[Optional]
InputVolume: (an existing file name)
    Volume containing the input grayscale data.
    argument: ``%s``, position: -2
OutputGeometry: (a boolean or a file name)
    Output that contains geometry model.
    argument: ``%s``, position: -1
decimate: (a float)
    Target reduction during decimation, as a decimal percentage
    reduction in the number of polygons. If 0, no decimation will be
    done.
    argument: ``--decimate %f``
name: (a unicode string)
    Name to use for this model.
    argument: ``--name %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
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.
    argument: ``--threshold %f``
pointnormals: (a boolean)
    Calculate the point normals? Calculated point normals make the
    surface appear smooth. Without point normals, the surface will
    appear faceted.
    argument: ``--pointnormals``
smooth: (an integer (int or long))
    Number of smoothing iterations. If 0, no smoothing will be done.
    argument: ``--smooth %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
splitnormals: (a boolean)
    Splitting normals is useful for visualizing sharp features. However
    it creates holes in surfaces which affect measurements
    argument: ``--splitnormals``
```

Outputs:

```
OutputGeometry: (an existing file name)
    Output that contains geometry model.
```

76.27.2 LabelMapSmoothing

[Link to code](#)

Wraps the executable 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:

```
[Optional]
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.
    argument: ``--labelToSmooth %d``
maxRMSError: (a float)
    The maximum RMS error.
    argument: ``--maxRMSError %f``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
numberOfIterations: (an integer (int or long))
    The number of iterations of the level set AntiAliasing algorithm
    argument: ``--numberOfIterations %d``
inputVolume: (an existing file name)
    Input label map to smooth
    argument: ``%s``, position: -2
gaussianSigma: (a float)
    The standard deviation of the Gaussian kernel
    argument: ``--gaussianSigma %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
outputVolume: (a boolean or a file name)
    Smoothed label map
    argument: ``%s``, position: -1
```

Outputs:

```
outputVolume: (an existing file name)
    Smoothed label map
```

76.27.3 MergeModels

[Link to code](#)

Wraps the executable 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:

```
[Optional]
Model2: (an existing file name)
    Model
    argument: ``%s``, position: -2
Model1: (an existing file name)
    Model
    argument: ``%s``, position: -3
ModelOutput: (a boolean or a file name)
    Model
    argument: ``%s``, position: -1
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a 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
```

Outputs:

```
ModelOutput: (an existing file name)
    Model
```

76.27.4 ModelMaker

[Link to code](#)

Wraps the executable 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>

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:

```

[Optional]
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.
    argument: ``--generateAll ``
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.
    argument: ``--end %d``
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.
    argument: ``--jointsmooth ``
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.
    argument: ``--name %s``
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).
    argument: ``--start %d``
pointnormals: (a boolean)
    Turn this flag on if you wish to calculate the normal vectors for
    the points.
    argument: ``--pointnormals ``
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.
    argument: ``--skipUnNamed ``
color: (an existing file name)
    Color table to make labels to colors and objects
    argument: ``--color %s``
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.
    argument: ``--filtertype %s``
environ: (a dictionary with keys which are a 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
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)
    argument: ``--debug ``

```

(continues on next page)

(continued from previous page)

```

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.
    argument: ``--labels %s``

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.
    argument: ``--decimate %f``

args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``

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.
    argument: ``--smooth %d``

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().
    argument: ``--saveIntermediateModels ``

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.
    argument: ``--modelSceneFile %s...``

splitnormals: (a boolean)
    Splitting normals is useful for visualizing sharp features. However
    it creates holes in surfaces which affects measurements.
    argument: ``--splitnormals ``

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.
    argument: ``--pad ``

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.
    argument: ``%s``, position: -1

```

Outputs:

```

modelSceneFile: (a list of items which are an existing file name)

```

(continues on next page)

(continued from previous page)

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.

76.27.5 ModelToLabelMap

[Link to code](#)

Wraps the executable 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:

```
[Optional]
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
distance: (a float)
    Sample distance
    argument: ``--distance %f``
InputVolume: (an existing file name)
    Input volume
    argument: ``%s``, position: -3
surface: (an existing file name)
    Model
    argument: ``%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
OutputVolume: (a boolean or a file name)
    The label volume
    argument: ``%s``, position: -1
```

Outputs:

```
OutputVolume: (an existing file name)
    The label volume
```

76.27.6 ProbeVolumeWithModel

[Link to code](#)

Wraps the executable 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:

```
[Optional]
OutputModel: (a boolean or a file name)
    Output 'painted' model
    argument: ``%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', nipype default value: {})
    Environment variables
InputVolume: (an existing file name)
    Volume to use to 'paint' the model
    argument: ``%s``, position: -3
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
InputModel: (an existing file name)
    Input model
    argument: ``%s``, position: -2
```

Outputs:

```
OutputModel: (an existing file name)
    Output 'painted' model
```

76.28 interfaces.slicer.utilities

76.28.1 EMSegmentTransformToNewFormat

[Link to code](#)

Wraps the executable command “EMSegmentTransformToNewFormat”.

title: Transform MRML Files to New EMSegmenter Standard

category: Utilities

description: Transform MRML Files to New EMSegmenter Standard

Inputs:

```
[Optional]
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.
    argument: ``--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
    argument: ``--outputMRMLFileName %s``
templateFlag: (a boolean)
    Set to true if the transformed mrml file should be used as template
    file
    argument: ``--templateFlag ``
```

(continues on next page)

(continued from previous page)

```
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
environ: (a dictionary with keys which are a 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
```

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
```

77.1 interfaces.spm.model

77.1.1 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
spm_mat_file: (an existing file name)
              Absolute path to SPM.mat
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)))
```

(continues on next page)

(continued from previous page)

```

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

[Optional]
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
matlab_cmd: (a unicode string)
matlab command to use
group_contrast: (a boolean)
higher level contrast
mutually_exclusive: use_derivs
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, nipyne default value: True)
Generate SPM8 and higher compatible jobs

```

Outputs:

```

spmF_images: (a list of items which are an existing file name)
stat images from an F-contrast
spm_mat_file: (an existing file name)
Updated SPM mat file
spmT_images: (a list of items which are an existing file name)
stat images from a t-contrast
ess_images: (a list of items which are an existing file name)
contrast images from an F-contrast
con_images: (a list of items which are an existing file name)
contrast images from a t-contrast

```

References:

None

77.1.2 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.estimate_method = {'Classical': 1}
>>> est.run()

```

Inputs:

```

[Mandatory]
spm_mat_file: (an existing file name)
    Absolute path to SPM.mat
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)

[Optional]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
write_residuals: (a boolean)
    Write individual residual images
use_mcr: (a boolean)
    Run m-code using SPM MCR
flags: (a dictionary with keys which are any value and with values
    which are any value)
    Additional arguments
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs

```

Outputs:

```

beta_images: (a list of items which are an existing, uncompressed
    file (valid extensions: [.img, .nii, .hdr]))
    design parameter estimates
SDbetas: (a list of items which are an existing, uncompressed file
    (valid extensions: [.img, .nii, .hdr]))
    Images of the standard deviation of parameter posteriors
spm_mat_file: (an existing file name)
    Updated SPM mat file
labels: (an existing, uncompressed file (valid extensions: [.img,
    .nii, .hdr]))
    label file
residual_image: (an existing, uncompressed file (valid extensions:
    [.img, .nii, .hdr]))
    Mean-squared image of the residuals
RPVimage: (an existing, uncompressed file (valid extensions: [.img,
    .nii, .hdr]))
    Resels per voxel image
ARcoef: (a list of items which are an existing, uncompressed file
    (valid extensions: [.img, .nii, .hdr]))
    Images of the AR coefficient
mask_image: (an existing, uncompressed file (valid extensions: [.img,
    .nii, .hdr]))
    binary mask to constrain estimation
Cbetas: (a list of items which are an existing, uncompressed file
    (valid extensions: [.img, .nii, .hdr]))
    Images of the parameter posteriors
residual_images: (a list of items which are an existing, uncompressed
    file (valid extensions: [.img, .nii, .hdr]))
    individual residual images (requires `write_residuals`
SDError: (a list of items which are an existing, uncompressed file
    (valid extensions: [.img, .nii, .hdr]))

```

(continues on next page)

(continued from previous page)

Images of the standard deviation of the error

References:

None

77.1.3 FactorialDesign[Link to code](#)

Base class for factorial designs

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=77>

Inputs:

```

[Optional]
mfile: (a boolean, nipyte default value: True)
    Run m-code using m-file
use_implicit_threshold: (a boolean)
    use implicit mask NaNs or zeros to threshold
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
threshold_mask_absolute: (a float)
    use an absolute threshold
    mutually_exclusive: threshold_mask_none, threshold_mask_relative
threshold_mask_relative: (a float)
    threshold using a proportion of the global value
    mutually_exclusive: threshold_mask_absolute, threshold_mask_none
explicit_mask_file: (a file name)
    use an implicit mask file to threshold
spm_mat_dir: (an existing directory name)
    directory to store SPM.mat file (opt)
threshold_mask_none: (a boolean)
    do not use threshold masking
    mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
matlab_cmd: (a unicode string)
    matlab command to use
global_normalization: (1 or 2 or 3)
    global normalization None-1, Proportional-2, ANCOVA-3
global_calc_values: (a list of items which are a float)
    omit global calculation
    mutually_exclusive: global_calc_mean, global_calc_omit
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}
no_grand_mean_scaling: (a boolean)
    do not perform grand mean scaling
global_calc_omit: (a boolean)
    omit global calculation
    mutually_exclusive: global_calc_mean, global_calc_values
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipyte default value: True)
    Generate SPM8 and higher compatible jobs
global_calc_mean: (a boolean)
    use mean for global calculation
    mutually_exclusive: global_calc_omit, global_calc_values

```


Outputs:

```
spm_mat_file: (an existing file name)
              SPM mat file
```

References:

None

77.1.4 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
session_info: (any value)
Session specific information generated by ``modelgen.SpecifyModel``
interscan_interval: (a float)
Interscan interval in secs
timing_units: ('secs' or 'scans')
units for specification of onsets
```

```
[Optional]
mfile: (a boolean, nipy default value: True)
Run m-code using m-file
volterra_expansion_order: (1 or 2)
Model interactions - yes:1, no:2
paths: (a list of items which are a directory name)
```

(continues on next page)

(continued from previous page)

```

    Paths to add to matlabpath
model_serial_correlations: ('AR(1)' or 'FAST' or 'none')
    Model serial correlations AR(1), FAST or none. FAST is available in
    SPM12
microtime_resolution: (an integer (int or long))
    Number of time-bins per scan in secs (opt)
global_intensity_normalization: ('none' or 'scaling')
    Global intensity normalization - scaling or none
microtime_onset: (a float)
    The onset/time-bin in seconds for alignment (opt)
spm_mat_dir: (an existing directory name)
    directory to store SPM.mat file (opt)
matlab_cmd: (a unicode string)
    matlab command to use
mask_image: (an existing file name)
    Image for explicitly masking the analysis
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)
mask_threshold: ('-Inf' or a float, nipy default value: -Inf)
    Thresholding for the mask
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

77.1.5 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]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file

```

(continues on next page)

(continued from previous page)

```

use_implicit_threshold: (a boolean)
    use implicit mask NaNs or zeros to threshold
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
include_intercept: (a boolean, nipy default value: True)
    Include intercept in design
threshold_mask_absolute: (a float)
    use an absolute threshold
    mutually_exclusive: threshold_mask_none, threshold_mask_relative
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}
explicit_mask_file: (a file name)
    use an implicit mask file to threshold
spm_mat_dir: (an existing directory name)
    directory to store SPM.mat file (opt)
threshold_mask_none: (a boolean)
    do not use threshold masking
    mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
matlab_cmd: (a unicode string)
    matlab command to use
global_normalization: (1 or 2 or 3)
    global normalization None-1, Proportional-2, ANCOVA-3
global_calc_values: (a list of items which are a float)
    omit global calculation
    mutually_exclusive: global_calc_mean, global_calc_omit
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}
no_grand_mean_scaling: (a boolean)
    do not perform grand mean scaling
global_calc_omit: (a boolean)
    omit global calculation
    mutually_exclusive: global_calc_mean, global_calc_values
threshold_mask_relative: (a float)
    threshold using a proportion of the global value
    mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
global_calc_mean: (a boolean)
    use mean for global calculation
    mutually_exclusive: global_calc_omit, global_calc_values

```

Outputs:

```

spm_mat_file: (an existing file name)
    SPM mat file

```

References:

None

77.1.6 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]
mfile: (a boolean, nipy default value: True)
       Run m-code using m-file
use_implicit_threshold: (a boolean)
                        use implicit mask NaNs or zeros to threshold
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
threshold_mask_absolute: (a float)
                        use an absolute threshold
                        mutually_exclusive: threshold_mask_none, threshold_mask_relative
global_normalization: (1 or 2 or 3)
                      global normalization None-1, Proportional-2, ANCOVA-3
explicit_mask_file: (a file name)
                   use an implicit mask file to threshold
spm_mat_dir: (an existing directory name)
             directory to store SPM.mat file (opt)
threshold_mask_none: (a boolean)
                    do not use threshold masking
                    mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
matlab_cmd: (a unicode string)
            matlab command to use
global_calc_values: (a list of items which are a float)
                   omit global calculation
                   mutually_exclusive: global_calc_mean, global_calc_omit
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}
no_grand_mean_scaling: (a boolean)
                      do not perform grand mean scaling
global_calc_omit: (a boolean)
                  omit global calculation
                  mutually_exclusive: global_calc_mean, global_calc_values
threshold_mask_relative: (a float)
                        threshold using a proportion of the global value
                        mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use_mcr: (a boolean)
         Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
              Generate SPM8 and higher compatible jobs
```

(continues on next page)

(continued from previous page)

```
global_calc_mean: (a boolean)
    use mean for global calculation
    mutually_exclusive: global_calc_omit, global_calc_values
```

Outputs:

```
spm_mat_file: (an existing file name)
    SPM mat file
```

References:

None

77.1.7 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]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
use_implicit_threshold: (a boolean)
    use implicit mask NaNs or zeros to threshold
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
threshold_mask_absolute: (a float)
    use an absolute threshold
    mutually_exclusive: threshold_mask_none, threshold_mask_relative
global_normalization: (1 or 2 or 3)
    global normalization None-1, Proportional-2, ANCOVA-3
explicit_mask_file: (a file name)
    use an implicit mask file to threshold
spm_mat_dir: (an existing directory name)
    directory to store SPM.mat file (opt)
threshold_mask_none: (a boolean)
    do not use threshold masking
    mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
matlab_cmd: (a unicode string)
    matlab command to use
ancova: (a boolean)
    Specify ancova-by-factor regressors
global_calc_values: (a list of items which are a float)
```

(continues on next page)

(continued from previous page)

```

        omit global calculation
        mutually_exclusive: global_calc_mean, global_calc_omit
    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}
    no_grand_mean_scaling: (a boolean)
        do not perform grand mean scaling
    global_calc_omit: (a boolean)
        omit global calculation
        mutually_exclusive: global_calc_mean, global_calc_values
    grand_mean_scaling: (a boolean)
        Perform grand mean scaling
    threshold_mask_relative: (a float)
        threshold using a proportion of the global value
        mutually_exclusive: threshold_mask_absolute, threshold_mask_none
    use_mcr: (a boolean)
        Run m-code using SPM MCR
    use_v8struct: (a boolean, nipy default value: True)
        Generate SPM8 and higher compatible jobs
    global_calc_mean: (a boolean)
        use mean for global calculation
        mutually_exclusive: global_calc_omit, global_calc_values

```

Outputs:

```

spm_mat_file: (an existing file name)
    SPM mat file

```

References:

None

77.1.8 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]
spm_mat_file: (an existing file name)
    absolute path to SPM.mat
contrast_index: (an integer (int or long))
    which contrast in the SPM.mat to use
stat_image: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

        stat image

[Optional]
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
height_threshold: (a float, nipy default value: 0.05)
        value for initial thresholding (defining clusters)
extent_fdr_p_threshold: (a float, nipy default value: 0.05)
        p threshold on FDR corrected cluster size probabilities
matlab_cmd: (a unicode string)
        matlab command to use
use_topo_fdr: (a boolean, nipy default value: True)
        whether to use FDR over cluster extent probabilities
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.
extent_threshold: (an integer (int or long), nipy default value: 0)
        Minimum cluster size in voxels
height_threshold_type: ('p-value' or 'stat', nipy default value:
        p-value)
        Is the cluster forming threshold a stat value or p-value?
use_v8struct: (a boolean, nipy default value: True)
        Generate SPM8 and higher compatible jobs
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)

```

Outputs:

```

n_clusters: (an integer (int or long))
cluster_forming_thr: (a float)
activation_forced: (a boolean)
thresholded_map: (an existing file name)
pre_topo_fdr_map: (an existing file name)
pre_topo_n_clusters: (an integer (int or long))

```

References:

None

77.1.9 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

```

(continues on next page)

(continued from previous page)

```
>>> thresh.inputs.height_threshold = 4.56
>>> thresh.run()
```

Inputs:

```
[Mandatory]
spm_mat_file: (an existing file name)
    absolute path to SPM.mat
contrast_index: (an integer (int or long))
    which contrast in the SPM.mat to use
stat_image: (an existing file name)
    stat image
height_threshold: (a float)
    stat value for initial thresholding (defining clusters)

[Optional]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
extent_threshold: (an integer (int or long), nipy default value: 0)
    Minimum cluster size in voxels
use_mcr: (a boolean)
    Run m-code using SPM MCR
```

Outputs:

```
clusterwise_P_FDR: (a float)
voxelwise_P_FDR: (a float)
voxelwise_P_RF: (a float)
voxelwise_P_Bonf: (a float)
voxelwise_P_uncor: (a float)
clusterwise_P_RF: (a float)
```

References:

None

77.1.10 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]
mfile: (a boolean, nipype default value: True)
       Run m-code using m-file
use_implicit_threshold: (a boolean)
       use implicit mask NaNs or zeros to threshold
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
unequal_variance: (a boolean)
       Are the variances equal or unequal between groups
threshold_mask_absolute: (a float)
       use an absolute threshold
       mutually_exclusive: threshold_mask_none, threshold_mask_relative
global_normalization: (1 or 2 or 3)
       global normalization None-1, Proportional-2, ANCOVA-3
explicit_mask_file: (a file name)
       use an implicit mask file to threshold
dependent: (a boolean)
       Are the measurements dependent between levels
threshold_mask_none: (a boolean)
       do not use threshold masking
       mutually_exclusive: threshold_mask_absolute, threshold_mask_relative
matlab_cmd: (a unicode string)
       matlab command to use
global_calc_values: (a list of items which are a float)
       omit global calculation
       mutually_exclusive: global_calc_mean, global_calc_omit
spm_mat_dir: (an existing directory name)
       directory to store SPM.mat file (opt)
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}
no_grand_mean_scaling: (a boolean)
       do not perform grand mean scaling
global_calc_omit: (a boolean)
       omit global calculation
       mutually_exclusive: global_calc_mean, global_calc_values
threshold_mask_relative: (a float)
       threshold using a proportion of the global value
       mutually_exclusive: threshold_mask_absolute, threshold_mask_none
use_mcr: (a boolean)
       Run m-code using SPM MCR
use_v8struct: (a boolean, nipype default value: True)
       Generate SPM8 and higher compatible jobs
global_calc_mean: (a boolean)
       use mean for global calculation
       mutually_exclusive: global_calc_omit, global_calc_values

```

Outputs:

```
spm_mat_file: (an existing file name)
              SPM mat file
```

References:

None

77.2 interfaces.spm.preprocess

77.2.1 ApplyDeformations

[Link to code](#)**Inputs:**

```
[Mandatory]
in_files: (a list of items which are an existing, uncompressed file
          (valid extensions: [.img, .nii, .hdr]))
deformation_field: (an existing file name)
reference_volume: (an existing, uncompressed file (valid extensions:
                  [.img, .nii, .hdr]))

[Optional]
mfile: (a boolean, nipy default value: True)
       Run m-code using m-file
matlab_cmd: (a unicode string)
            matlab command to use
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
interp: (0 <= a long integer <= 7)
        degree of b-spline used for interpolation
```

Outputs:

```
out_files: (a list of items which are an existing file name)
```

References:

None

77.2.2 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'
```

(continues on next page)

(continued from previous page)

```
>>> coreg.inputs.source = 'structural.nii'
>>> coreg.run()
```

Inputs:

```
[Mandatory]
source: (a list of items which are an existing, uncompressed file
        (valid extensions: [.img, .nii, .hdr]))
        file to register to target
target: (an existing, uncompressed file (valid extensions: [.img,
        .nii, .hdr]))
        reference file to register to

[Optional]
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
mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
write_mask: (a boolean)
             True/False mask output image
tolerance: (a list of items which are a float)
            acceptable tolerance for each of 12 params
write_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
use_mcr: (a boolean)
          Run m-code using SPM MCR
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]
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
matlab_cmd: (a unicode string)
             matlab command to use
out_prefix: (a string, nipy default value: r)
            coregistered output prefix
apply_to_files: (a list of items which are an existing file name)
                files to apply transformation to
separation: (a list of items which are a float)
            sampling separation in mm
use_v8struct: (a boolean, nipy default value: True)
               Generate SPM8 and higher compatible jobs
fwhm: (a list of from 2 to 2 items which are a float)
       gaussian smoothing kernel width (mm)
```

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

77.2.3 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 nipyne.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]
image_files: (a list of items which are an existing, uncompressed
             file (valid extensions: [.img, .nii, .hdr]))
             A list of files to be warped
flowfield_files: (a list of items which are an existing, uncompressed
                 file (valid extensions: [.img, .nii, .hdr]))
                 DARTEL flow fields u_rcl*

[Optional]
mfile: (a boolean, nipyne default value: True)
       Run m-code using m-file
matlab_cmd: (a unicode string)
            matlab command to use
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
use_v8struct: (a boolean, nipyne default value: True)
              Generate SPM8 and higher compatible jobs
modulate: (a boolean)
           Modulate images
use_mcr: (a boolean)
          Run m-code using SPM MCR
iterations: (0 <= a long integer <= 9)
            The number of iterations: log2(number of time steps)
interp: (0 <= a long integer <= 7)
         degree of b-spline used for interpolation
```

Outputs:

```
warped_files: (a list of items which are an existing file name)
```

References:

None

77.2.4 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 nipyre.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, uncompressed file (valid extensions: [.img, .nii,
              .hdr]))
              A list of files to be segmented

[Optional]
template_prefix: (a unicode string, nipyre default value: Template)
                 Prefix for template
mfile: (a boolean, nipyre default value: True)
       Run m-code using m-file
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
matlab_cmd: (a unicode string)
            matlab command to use
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
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
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:

```
final_template_file: (an existing file name)
                    final DARTEL template
dartel_flow_fields: (a list of items which are an existing file name)
                   DARTEL flow fields
template_files: (a list of items which are an existing file name)
                Templates from different stages of iteration
```

References:

None

77.2.5 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]
flowfield_files: (a list of items which are an existing, uncompressed
                  file (valid extensions: [.img, .nii, .hdr]))
                  DARTEL flow fields u_rcl*
template_file: (an existing, uncompressed file (valid extensions:
                [.img, .nii, .hdr]))
                DARTEL template
apply_to_files: (a list of items which are an existing, uncompressed
                file (valid extensions: [.img, .nii, .hdr]))
                Files to apply the transform to

[Optional]
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
bounding_box: (a tuple of the form: (a float, a float, a float, a
                                     float, a float, a float))
               Voxel sizes for output file
voxel_size: (a tuple of the form: (a float, a float, a float))
             Voxel sizes for output file
matlab_cmd: (a unicode string)
             matlab command to use
modulate: (a boolean)
           Modulate out images - no modulation preserves concentrations
use_v8struct: (a boolean, nipy default value: True)
              Generate SPM8 and higher compatible jobs
fwhm: (a list of from 3 to 3 items which are a float or a float)
       3-list of fwhm for each dimension
```

Outputs:

```
normalized_files: (a list of items which are an existing file name)
                  Normalized files in MNI space
```

(continues on next page)

(continued from previous page)

```
normalization_parameter_file: (an existing file name)
    Transform parameters to MNI space
```

References:

None

77.2.6 FieldMap[Link to code](#)

Use the fieldmap toolbox from spm to calculate the voxel displacement map (VDM).

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=173>**To do**

Deal with real/imag magnitude images and with the two phase files case.

Examples

```
>>> from nipy.interfaces.spm import FieldMap
>>> fm = FieldMap()
>>> fm.inputs.phase_file = 'phase.nii'
>>> fm.inputs.magnitude_file = 'magnitude.nii'
>>> fm.inputs.echo_times = (5.19, 7.65)
>>> fm.inputs.blip_direction = 1
>>> fm.inputs.total_readout_time = 15.6
>>> fm.inputs.epi_file = 'epi.nii'
>>> fm.run()
```

Inputs:

```
[Mandatory]
phase_file: (an existing file name)
    presubtracted phase file
total_readout_time: (a float)
    total EPI readout time
epi_file: (an existing file name)
    EPI to unwarp
magnitude_file: (an existing file name)
    presubtracted magnitude file
blip_direction: (1 or -1)
    polarity of the phase-encode blips
echo_times: (a tuple of the form: (a float, a float))
    short and long echo times

[Optional]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
pad: (a long integer >= 0, nipy default value: 0)
    padding kernel width
writeunwarped: (a boolean, nipy default value: False)
    write unwarped EPI
matchvdm: (a boolean, nipy default value: True)
    match VDM to EPI
nerode: (a long integer >= 0, nipy default value: 2)
```

(continues on next page)

(continued from previous page)

```

        number of erosions
matlab_cmd: (a unicode string)
        matlab command to use
reg: (a float, nipy default value: 0.02)
        regularization value used in the segmentation
ndilate: (a long integer >= 0, nipy default value: 4)
        number of erosions
use_v8struct: (a boolean, nipy default value: True)
        Generate SPM8 and higher compatible jobs
sessname: (a unicode string, nipy default value: _run-)
        VDM filename extension
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
mask_fwhm: (a long integer >= 0, nipy default value: 5)
        gaussian smoothing kernel width
anat_file: (an existing file name)
        anatomical image for comparison
thresh: (a float, nipy default value: 0.5)
        threshold used to create brain mask from segmented data
jobtype: ('calculatevdm' or 'applyvdm', nipy default value:
        calculatevdm)
        one of: calculatevdm, applyvdm
matchanat: (a boolean, nipy default value: True)
        match anatomical image to EPI
ws: (a boolean, nipy default value: True)
        weighted smoothing
unwarp_fwhm: (a long integer >= 0, nipy default value: 10)
        gaussian smoothing kernel width
epifm: (a boolean, nipy default value: False)
        epi-based field map
method: ('Mark3D' or 'Mark2D' or 'Huttonish', nipy default value:
        Mark3D)
        One of: Mark3D, Mark2D, Huttonish
jacobian_modulation: (a boolean, nipy default value: False)
        jacobian modulation
template: (an existing file name)
        template image for brain masking
use_mcr: (a boolean)
        Run m-code using SPM MCR
maskbrain: (a boolean, nipy default value: True)
        masking or no masking of the brain

```

Outputs:

```

vdm: (an existing file name)
        voxel difference map

```

References:

None

77.2.7 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, uncompressed
                file (valid extensions: [.img, .nii, .hdr]))
                A list of files to be segmented

[Optional]
affine_regularization: ('mni' or 'eastern' or 'subj' or 'none')
                        mni, eastern, subj, none
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
write_deformation_fields: (a list of from 2 to 2 items which are a
                           boolean)
                           Which deformation fields to write:[Inverse, Forward]
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, uncompressed file (valid extensions:
                 [.img, .nii, .hdr]), 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
matlab_cmd: (a unicode string)
             matlab command to use
warping_regularization: (a list of from 5 to 5 items which are a
                        float or a float)
```

(continues on next page)

(continued from previous page)

```

        Warping regularization parameter(s). Accepts float or list of floats
        (the latter is required by SPM12)
    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 reguralisation (0-10)
        - FWHM of Gaussian smoothness of bias
        - which maps to save (Field, Corrected) - a tuple of two boolean
        values
    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:

```

transformation_mat: (a list of items which are an existing file name)
    Normalization transformation
inverse_deformation_field: (a list of items which are an existing
    file name)
normalized_class_images: (a list of items which are a list of items
    which are an existing file name)
    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
dartel_input_images: (a list of items which are a list of items which
    are an existing file name)
    dartel imported class images
modulated_class_images: (a list of items which are a list of items
    which are an existing file name)
    modulated+normalized class images
bias_corrected_images: (a list of items which are an existing file
    name)
    bias corrected images
forward_deformation_field: (a list of items which are an existing
    file name)
bias_field_images: (a list of items which are an existing file name)
    bias field images

```

References:

None

77.2.8 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, uncompressed file
    (valid extensions: [.img, .nii, .hdr]))
    file to normalize to template
    mutually_exclusive: parameter_file
template: (an existing file name)
    template file to normalize to
    mutually_exclusive: parameter_file

[Optional]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
template_image_smoothing: (a float)
    template smoothing
affine_regularization_type: ('mni' or 'size' or 'none')
    mni, size, none
write_voxel_sizes: (a list of from 3 to 3 items which are a float)
    3-element list
source_weight: (a file name)
    name of weighting image for source
matlab_cmd: (a unicode string)
    matlab command to use
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
DCT_period_cutoff: (a float)
    Cutoff of for DCT bases
write_interp: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
write_preserve: (a boolean)
    True/False warped images are modulated
jobtype: ('estwrite' or 'est' or 'write', nipy default value:
    estwrite)
    Estimate, Write or do both
nonlinear_iterations: (an integer (int or long))
    Number of iterations of nonlinear warping
use_mcr: (a boolean)
    Run m-code using SPM MCR
nonlinear_regularization: (a float)
    the amount of the regularization for the nonlinear part of the
    normalization
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
out_prefix: (a string, nipy default value: w)
    normalized output prefix
template_weight: (a file name)
    name of weighting image for template
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_wrap: (a list of items which are an integer (int or long))

```

(continues on next page)

(continued from previous page)

```

    Check if interpolation should wrap in [x,y,z] - list of bools
    source_image_smoothing: (a float)
        source smoothing

```

Outputs:

```

normalized_files: (a list of items which are an existing file name)
    Normalized other files
normalization_parameters: (a list of items which are an existing file
    name)
    MAT files containing the normalization parameters
normalized_source: (a list of items which are an existing file name)
    Normalized source files

```

References:

None

77.2.9 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]
image_to_align: (an existing, uncompressed file (valid extensions:
    [.img, .nii, .hdr]))
    file to estimate normalization parameters with
    mutually_exclusive: deformation_file
deformation_file: (a uncompressed file (valid extensions: [.img,
    .nii, .hdr]))
    file y_*.nii containing 3 deformation fields for the deformation in
    x, y and z dimension
    mutually_exclusive: image_to_align, tpm

```

```

[Optional]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
write_interp: (0 <= a long integer <= 7)
    degree of b-spline used for interpolation
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

```

(continues on next page)

(continued from previous page)

```

affine_regularization_type: ('mni' or 'size' or 'none')
    mni, size, none
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
jobtype: ('estwrite' or 'est' or 'write', nipy default value:
    estwrite)
    Estimate, Write or do Both
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
apply_to_files: (a list of items which are an existing, uncompressed
    file (valid extensions: [.img, .nii, .hdr]) or a list of items
    which are an existing, uncompressed file (valid extensions: [.img,
    .nii, .hdr]))
    files to apply transformation to
matlab_cmd: (a unicode string)
    matlab command to use
tpm: (an existing file name)
    template in form of tissue probability maps to normalize to
    mutually_exclusive: deformation_file
warping_regularization: (a list of from 5 to 5 items which are a
    float)
    controls balance between parameters and data
out_prefix: (a string, nipy default value: w)
    Normalized output prefix
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
smoothness: (a float)
    value (in mm) to smooth the data before normalization
use_mcr: (a boolean)
    Run m-code using SPM MCR
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)
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

```

Outputs:

```

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
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

```

References:

None

77.2.10 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 an existing, uncompressed file
          (valid extensions: [.img, .nii, .hdr]) or a list of items which
          are an existing, uncompressed file (valid extensions: [.img, .nii,
          .hdr]))
          list of filenames to realign

[Optional]
mfile: (a boolean, nipy default value: True)
       Run m-code using m-file
write_interp: (0 <= a long integer <= 7)
              degree of b-spline used for interpolation
weight_img: (an existing file name)
             filename of weighting image
jobtype: ('estwrite' or 'estimate' or 'write', nipy default value:
          estwrite)
          one of: estimate, write, estwrite
use_mcr: (a boolean)
          Run m-code using SPM MCR
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_mask: (a boolean)
            True/False mask output image
register_to_mean: (a boolean)
                 Indicate whether realignment is done to the mean image
interp: (0 <= a long integer <= 7)
        degree of b-spline used for interpolation
matlab_cmd: (a unicode string)
            matlab command to use
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
out_prefix: (a string, nipy default value: r)
            realigned output prefix
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]
quality: (0.0 <= a floating point number <= 1.0)
          0.1 = fast, 1.0 = precise
separation: (a floating point number >= 0.0)
            sampling separation in mm
write_which: (a list of items which are a value of class 'int',
             nipy default value: [2, 1])
             determines which images to reslice
use_v8struct: (a boolean, nipy default value: True)
```

(continues on next page)

(continued from previous page)

```

        Generate SPM8 and higher compatible jobs
    fwhm: (a floating point number >= 0.0)
        gaussian smoothing kernel width

```

Outputs:

```

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.
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.
mean_image: (an existing file name)
    Mean image file from the realignment
realignment_parameters: (a list of items which are an existing file
    name)
    Estimated translation and rotation parameters

```

References:

None

77.2.11 RealignUnwarp[Link to code](#)

Use `spm_uw_estimate` for estimating within subject registration and unwarping of time series. Function accepts only one single field map. If `in_files` is a list of files they will be treated as separate sessions but associated to the same fieldmap.

<http://www.fil.ion.ucl.ac.uk/spm/doc/manual.pdf#page=31>

Examples

```

>>> import nipy.interfaces.spm as spm
>>> realignUnwarp = spm.RealignUnwarp()
>>> realignUnwarp.inputs.in_files = ['functional.nii', 'functional2.nii']
>>> realignUnwarp.inputs.phase_map = 'voxeldisplacemap.vdm'
>>> realignUnwarp.inputs.register_to_mean = True
>>> realignUnwarp.run()

```

Inputs:

```

[Mandatory]
in_files: (a list of items which are an existing, uncompressed file
    (valid extensions: [.img, .nii, .hdr]) or a list of items which
    are an existing, uncompressed file (valid extensions: [.img, .nii,
    .hdr]))
    list of filenames to realign and unwarp

[Optional]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
weight_img: (an existing file name)

```

(continues on next page)

(continued from previous page)

```

        filename of weighting image
est_second_order_effects: (a list of from 1 to 6 items which are an
        integer (int or long))
        List of second order terms to model second derivatives of.
est_num_of_iterations: (a list of items which are a value of class
        'int', nipyte default value: [5])
        Number of iterations.
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
register_to_mean: (a boolean)
        Indicate whether realignment is done to the mean image
interp: (0 <= a long integer <= 7)
        degree of b-spline used for interpolation
matlab_cmd: (a unicode string)
        matlab command to use
out_prefix: (a string, nipyte default value: u)
        realigned and unwrapped output prefix
reslice_mask: (a boolean)
        True/False mask output image
quality: (0.0 <= a floating point number <= 1.0)
        0.1 = fast, 1.0 = precise
est_unwarp_fwhm: (a floating point number >= 0.0)
        gaussian smoothing kernel width for unwarp
est_re_est_mov_par: (a boolean)
        Re-estimate movement parameters at each unwarping iteration.
use_v8struct: (a boolean, nipyte default value: True)
        Generate SPM8 and higher compatible jobs
est_basis_func: (a list of from 2 to 2 items which are an integer
        (int or long))
        Number of basis functions to use for each dimension
reslice_which: (a list of items which are a value of class 'int',
        nipyte default value: [2, 1])
        determines which images to reslice
est_jacobian_deformations: (a boolean)
        Jacobian deformations. In theory a good idea to include them, in
        practice a bad idea. Default: No.
phase_map: (a file name)
        Voxel displacement map to use in unwarping. Unlike SPM standard
        behaviour, the same map will be used for all sessions
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]
est_first_order_effects: (a list of from 1 to 6 items which are an
        integer (int or long))
        First order effects should only depend on pitch and roll, i.e. [4 5]
reslice_interp: (0 <= a long integer <= 7)
        degree of b-spline used for interpolation
est_reg_factor: (a list of items which are a value of class 'int',
        nipyte default value: [100000])
        Regularisation factor. Default: 100000 (medium).
separation: (a floating point number >= 0.0)
        sampling separation in mm
use_mcr: (a boolean)
        Run m-code using SPM MCR
fwhm: (a floating point number >= 0.0)
        gaussian smoothing kernel width
reslice_wrap: (a list of from 3 to 3 items which are an integer (int

```

(continues on next page)

(continued from previous page)

```

        or long))
    Check if interpolation should wrap in [x,y,z]
est_reg_order: (0 <= a long integer <= 3)
    This parameter determines how to balance the compromise between
    likelihood maximization and smoothness maximization of the estimated
    field.
est_taylor_expansion_point: (a string, nipy default value: Average)
    Point in position space to perform Taylor-expansion around.

```

Outputs:

```

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.
mean_image: (an existing file name)
    Mean image file from the realignment & unwarping
realignment_parameters: (a list of items which are an existing file
    name)
    Estimated translation and rotation parameters
realigned_unwarped_files: (a list of items which are a list of items
    which are an existing file name or an existing file name)
    Realigned and unwarped files written to disc.

```

References:

None

77.2.12 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, uncompressed file
    (valid extensions: [.img, .nii, .hdr]))
    one scan per subject

[Optional]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
affine_regularization: ('mni' or 'eastern' or 'subj' or 'none' or '')
    Possible options: "mni", "eastern", "subj", "none" (no
    regularisation), "" (no affine registration)
paths: (a list of items which are a directory name)

```

(continues on next page)

(continued from previous page)

```

    Paths to add to matlabpath
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]
sampling_distance: (a float)
    Sampling distance on data for parameter estimation
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]
use_mcr: (a boolean)
    Run m-code using SPM MCR
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]
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
matlab_cmd: (a unicode string)
    matlab command to use
warping_regularization: (a float)
    Controls balance between parameters and data
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)
mask_image: (an existing file name)
    Binary image to restrict parameter estimation
gaussians_per_class: (a list of items which are an integer (int or
    long))
    num Gaussians capture intensity distribution
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
tissue_prob_maps: (a list of items which are an existing file name)

```

(continues on next page)

(continued from previous page)

```

    list of gray, white & csf prob. (opt,)
warp_frequency_cutoff: (a float)
    Cutoff of DCT bases
save_bias_corrected: (a boolean)
    True/False produce a bias corrected image

```

Outputs:

```

transformation_mat: (an existing file name)
    Normalization transformation
normalized_csf_image: (a file name)
    normalized csf probability map
native_gm_image: (a file name)
    native space grey probability map
modulated_input_image: (a file name)
    bias-corrected version of input image
native_csf_image: (a file name)
    native space csf probability map
modulated_wm_image: (a file name)
    modulated, normalized white probability map
normalized_wm_image: (a file name)
    normalized white probability map
bias_corrected_image: (a file name)
    bias-corrected version of input image
inverse_transformation_mat: (an existing file name)
    Inverse normalization info
normalized_gm_image: (a file name)
    normalized grey probability map
modulated_csf_image: (a file name)
    modulated, normalized csf probability map
modulated_gm_image: (a file name)
    modulated, normalized grey probability map
native_wm_image: (a file name)
    native space white probability map

```

References:

None

77.2.13 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]
num_slices: (an integer (int or long))
    number of slices in a volume
time_acquisition: (a float)
    time of volume acquisition. usuallycalculated as TR-(TR/num_slices)
slice_order: (a list of items which are a float)
    1-based order or onset (in ms) in which slices are acquired
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)
in_files: (a list of items which are a list of items which are an
    existing, uncompressed file (valid extensions: [.img, .nii, .hdr])
    or an existing, uncompressed file (valid extensions: [.img, .nii,
    .hdr]))
    list of filenames to apply slice timing
time_repetition: (a float)
    time between volume acquisitions(start to start time)

[Optional]
mfile: (a boolean, nipyte 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
matlab_cmd: (a unicode string)
    matlab command to use
out_prefix: (a string, nipyte default value: a)
    slicetimed output prefix
use_v8struct: (a boolean, nipyte 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

77.2.14 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 nipyte.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, uncompressed file
          (valid extensions: [.img, .nii, .hdr]))
          list of files to smooth

[Optional]
mfile: (a boolean, nipy default value: True)
       Run m-code using m-file
matlab_cmd: (a unicode string)
            matlab command to use
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
out_prefix: (a string, nipy default value: s)
            smoothed output prefix
use_mcr: (a boolean)
         Run m-code using SPM MCR
fwhm: (a list of from 3 to 3 items which are a float or a float)
      3-list of fwhm for each dimension
implicit_masking: (a boolean)
                  A mask implied by a particular voxel value
use_v8struct: (a boolean, nipy default value: True)
              Generate SPM8 and higher compatible jobs
data_type: (an integer (int or long))
           Data type of the output images
```

Outputs:

```
smoothed_files: (a list of items which are an existing file name)
                smoothed files
```

References:

None

77.2.15 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, uncompressed file
          (valid extensions: [.img, .nii, .hdr]))
          A list of files to be segmented

[Optional]
mfile: (a boolean, nipy default value: True)
       Run m-code using m-file
wm_dartel: (0 <= a long integer <= 2, nipy default value: 0)
           0=None,1=rigid(SPM8 default),2=affine
gm_normalized: (a boolean, nipy default value: False)
use_sanlm_denoising_filter: (0 <= a long integer <= 2, nipy default
                             value: 2)
                             0=No denoising, 1=denoising,2=denoising multi-threaded
jacobian_determinant: (a boolean, nipy default value: False)
sampling_distance: (a float, nipy default value: 3)
                   Sampling distance on data for parameter estimation
wm_normalized: (a boolean, nipy default value: False)
mrf_weighting: (a float, nipy default value: 0.15)
wm_modulated_normalized: (0 <= a long integer <= 2, nipy default
                          value: 2)
                          0=none,1=affine+non-linear(SPM8 default),2=non-linear only
spatial_normalization: ('high' or 'low', nipy default value: high)
display_results: (a boolean, nipy default value: True)
deformation_field: (a tuple of the form: (a boolean, a boolean),
                    nipy default value: (0, 0))
                    forward and inverse field
pve_label_dartel: (0 <= a long integer <= 2, nipy default value: 0)
                  0=None,1=rigid(SPM8 default),2=affine
gm_native: (a boolean, nipy default value: False)
gm_modulated_normalized: (0 <= a long integer <= 2, nipy default
                          value: 2)
                          0=none,1=affine+non-linear(SPM8 default),2=non-linear only
use_v8struct: (a boolean, nipy default value: True)
               Generate SPM8 and higher compatible jobs
pve_label_native: (a boolean, nipy default value: False)
cleanup_partitions: (an integer (int or long), nipy default value:
                    1)
                    0=None,1=light,2=thorough
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
csf_native: (a boolean, nipy default value: False)
bias_corrected_affine: (a boolean, nipy default value: False)
matlab_cmd: (a unicode string)
             matlab command to use
wm_native: (a boolean, nipy default value: False)
tissues: (an existing, uncompressed file (valid extensions: [.img,
                    .nii, .hdr]))
          tissue probability map
gm_dartel: (0 <= a long integer <= 2, nipy default value: 0)
           0=None,1=rigid(SPM8 default),2=affine
csf_dartel: (0 <= a long integer <= 2, nipy default value: 0)
            0=None,1=rigid(SPM8 default),2=affine
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
dartel_template: (an existing, uncompressed file (valid extensions:

```

(continues on next page)

(continued from previous page)

```

        [.img, .nii, .hdr]))
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)
pve_label_normalized: (a boolean, nipy default value: False)
warping_regularization: (a float, nipy default value: 4)
    Controls balance between parameters and data
use_mcr: (a boolean)
    Run m-code using SPM MCR
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
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_normalized: (a boolean, nipy default value: False)
bias_corrected_normalized: (a boolean, nipy default value: True)
bias_corrected_native: (a boolean, nipy default value: False)

```

Outputs:

```

transformation_mat: (a list of items which are an existing file name)
    Normalization transformation
inverse_deformation_field: (a list of items which are an existing
    file name)
normalized_class_images: (a list of items which are a list of items
    which are an existing file name)
    normalized class images
bias_corrected_images: (a list of items which are an existing file
    name)
    bias corrected images
forward_deformation_field: (a list of items which are an existing
    file name)
pve_label_native_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)
dartel_input_images: (a list of items which are a list of items which
    are an existing file name)
    dartel imported class images
pve_label_normalized_images: (a list of items which are an existing
    file name)
normalized_bias_corrected_images: (a list of items which are an
    existing file name)
    bias corrected images
native_class_images: (a list of items which are a list of items which
    are an existing file name)
    native space probability maps
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

```

References:

None

77.3 interfaces.spm.utils

77.3.1 Analyze2nii

[Link to code](#)

Inputs:

```
[Mandatory]
analyze_file: (an existing file name)

[Optional]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
matlab_cmd: (a unicode string)
    matlab command to use
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:

```
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
nifti_file: (an existing file name)
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
```

References:

None

77.3.2 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 nipy.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]
mfile: (a boolean, nipy default value: True)
       Run m-code using m-file
deformation: (an existing file name)
             SN SPM deformation file
             mutually_exclusive: deformation_field
voxel_sizes: (a list of from 3 to 3 items which are a float)
             3-element list (opt)
use_mcr: (a boolean)
         Run m-code using SPM MCR
bounding_box: (a list of from 6 to 6 items which are a float)
              6-element list (opt)
interpolation: (0 <= a long integer <= 7)
               degree of b-spline used for interpolation
matlab_cmd: (a unicode string)
            matlab command to use
paths: (a list of items which are a directory name)
       Paths to add to matlabpath
deformation_field: (an existing file name)
                  SN SPM deformation file
                  mutually_exclusive: deformation
use_v8struct: (a boolean, nipy default value: True)
              Generate SPM8 and higher compatible jobs
target: (an existing file name)
        File defining target space
```

Outputs:

```
out_files: (a list of items which are an existing file name)
           Transformed files
```

References:

None

77.3.3 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]
mfile: (a boolean, nipy default value: True)
      Run m-code using m-file
matlab_cmd: (a unicode string)
           matlab command to use
paths: (a list of items which are a directory name)
      Paths to add to matlabpath
out_file: (a file name)
         output file name for transformed data
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

77.3.4 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 nipy.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 saves 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]
target: (an existing file name)
       target for generating affine transform
moving: (an existing file name)
```

(continues on next page)

(continued from previous page)

```

        volume transform can be applied to register with target

[Optional]
mfile: (a boolean, nipy default value: True)
        Run m-code using m-file
invmat: (a file name)
        Filename used to store inverse affine matrix
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
mat: (a file name)
        Filename used to store affine matrix
use_v8struct: (a boolean, nipy default value: True)
        Generate SPM8 and higher compatible jobs
matlab_cmd: (a unicode string)
        matlab command to use
use_mcr: (a boolean)
        Run m-code using SPM MCR

```

Outputs:

```

invmat: (a file name)
        Matlab file holding inverse transform
mat: (an existing file name)
        Matlab file holding transform

```

References:

None

77.3.5 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]
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.
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
icedims: (a boolean, nipy default value: False)

```

(continues on next page)

(continued from previous page)

```

    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.
format: ('nii' or 'img', nipy default value: nii)
    output format.
matlab_cmd: (a unicode string)
    matlab command to use
use_mcr: (a boolean)
    Run m-code using SPM MCR
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.
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

77.3.6 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]
mfile: (a boolean, nipy default value: True)
    Run m-code using m-file
matlab_cmd: (a unicode string)
    matlab command to use
paths: (a list of items which are a directory name)
    Paths to add to matlabpath
out_file: (a file name)
    Optional file to save resliced volume
use_mcr: (a boolean)
    Run m-code using SPM MCR
use_v8struct: (a boolean, nipy default value: True)
    Generate SPM8 and higher compatible jobs
interp: (0 <= a long integer <= 7, nipy default value: 0)
    degree of b-spline used for interpolation0 is nearest neighbor
    (default)

```

Outputs:

```

out_file: (an existing file name)
    resliced volume

```

References:

None

77.3.7 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]
mfile: (a boolean, nipy default value: True)
       Run m-code using m-file
matlab_cmd: (a unicode string)
            matlab command to use
interpolation: (0 <= a long integer <= 7)
              degree of b-spline used for interpolation
bounding_box: (a list of from 6 to 6 items which are a float)
              6-element list (opt)
use_v8struct: (a boolean, nipy default value: True)
              Generate SPM8 and higher compatible jobs
use_mcr: (a boolean)
          Run m-code using SPM MCR
voxel_sizes: (a list of from 3 to 3 items which are a float)
              3-element list (opt)
paths: (a list of items which are a directory name)
        Paths to add to matlabpath
target: (an existing file name)
        File defining target space
```

Outputs:

```
out_files: (a list of items which are an existing file name)
           Transformed files
```

References:

None

78.1 interfaces.utility.base

78.1.1 AssertEqual

[Link to code](#)

Inputs:

```
[Mandatory]
volume1: (an existing file name)
volume2: (an existing file name)
```

Outputs:

```
None
```

78.1.2 IdentityInterface

[Link to code](#)

Basic interface class generates identity mappings

Examples

```
>>> from nipy.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

78.1.3 Merge

[Link to code](#)

Basic interface class to merge inputs into a single list

Merge (1) will merge a list of lists

Examples

```
>>> from nipyre.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:

[Optional]
no_flatten: (a boolean, nipyre default value: **False**)
append to outlist instead of extending **in** vstack mode

(continues on next page)

(continued from previous page)

```
ravel_inputs: (a boolean, nipyne default value: False)
    ravel inputs when no_flatten is False
axis: ('vstack' or 'hstack', nipyne default value: vstack)
    direction in which to merge, hstack requires same number of elements
    in each input
```

Outputs:

```
out: (a list of items which are any value)
    Merged output
```

78.1.4 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 nipyne.interfaces.utility import Rename
>>> rename1 = Rename()
>>> rename1.inputs.in_file = os.path.join(datadir, "zstat1.nii.gz") # datadir is
↳ a directory with exemplary files, defined in conftest.py
>>> 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 = os.path.join(datadir, "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 = os.path.join(datadir, "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]
in_file: (an existing file name)
    file to rename
```

(continues on next page)

(continued from previous page)

```
format_string: (a unicode string)
    Python formatting string for output template

[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, nipyre default value: False)
    Use full path as input to regex parser
```

Outputs:

```
out_file: (a file name)
    softlink to original file with new name
```

78.1.5 Select

[Link to code](#)

Basic interface class to select specific elements from a list

Examples

```
>>> from nipyre.interfaces.utility import Select
>>> sl = Select()
>>> _ = sl.inputs.trait_set(inlist=[1, 2, 3, 4, 5], index=[3])
>>> out = sl.run()
>>> out.outputs.out
~
```

```
>>> _ = sl.inputs.trait_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
```

Outputs:

```
out: (a list of items which are any value)
    list of selected values
```

78.1.6 Split

[Link to code](#)

Basic interface class to split lists into multiple outputs

Examples

```
>>> from nipyype.interfaces.utility import Split
>>> sp = Split()
>>> _ = sp.inputs.trait_set(inlist=[1, 2, 3], splits=[2, 1])
>>> out = sp.run()
>>> out.outputs.out1
[1, 2]
```

Inputs:

```
[Mandatory]
splits: (a list of items which are an integer (int or long))
        Number of outputs in each split - should add to number of inputs
inlist: (a list of items which are any value)
        list of values to split

[Optional]
squeeze: (a boolean, nipyype default value: False)
         unfold one-element splits removing the list
```

Outputs:

None

78.2 interfaces.utility.csv

78.2.1 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, nipyre default value: False)
        True if the first line is a column header
```

Outputs:

```
None
```

78.3 interfaces.utility.wrappers

78.3.1 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
```

Outputs:

```
None
```

79.1 interfaces.vista.vista

79.1.1 Vnifti2Image

[Link to code](#)

Wraps the executable 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
         argument: ``-in %s``, position: 1

[Optional]
attributes: (an existing file name)
            attribute file
            argument: ``-attr %s``, position: 2
out_file: (a file name)
          output data file
          argument: ``-out %s``, position: -1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a 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: {})
```

(continues on next page)

(continued from previous page)

Environment variables

Outputs:

```
out_file: (an existing file name)
          Output vista file
```

79.1.2 VtoMat[Link to code](#)Wraps the executable 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
         argument: ``-in %s``, position: 1

[Optional]
out_file: (a file name)
         output mat file
         argument: ``-out %s``, position: -1
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
```

Outputs:

```
out_file: (an existing file name)
          Output mat file
```

80.1 interfaces.workbench.base

80.1.1 WBCommand

[Link to code](#)

Base support for workbench commands.

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
```

Outputs:

None

80.2 interfaces.workbench.metric

80.2.1 MetricResample

[Link to code](#)

Wraps the executable command `wb_command -metric-resample`.

Resample a metric file to a different mesh

Resamples a metric file, given two spherical surfaces that are in register. If `ADAP_BARY_AREA` is used, exactly one of `-area-surfs` or `-area-metrics` must be specified.

The `ADAP_BARY_AREA` method is recommended for ordinary metric data, because it should use all data while downsampling, unlike `BARYCENTRIC`. The recommended areas option for most data is individual midthicknesses for individual data, and averaged vertex area metrics from individual midthicknesses for group average data.

The `-current-roi` option only masks the input, the output may be slightly dilated in comparison, consider

using `-metric-mask` on the output when using `-current-roi`.

The `-largest` option results in nearest vertex behavior when used with BARYCENTRIC. When resampling a binary metric, consider thresholding at 0.5 after resampling rather than using `-largest`.

```
>>> from nipy.interfaces.workbench import MetricResample
>>> metres = MetricResample()
>>> metres.inputs.in_file = 'sub-01_task-rest_bold_space-fsaverage5.L.func.gii'
>>> metres.inputs.method = 'ADAP_BARY_AREA'
>>> metres.inputs.current_sphere = 'fsaverage5_std_sphere.L.10k_fsavg_L.surf.gii'
>>> metres.inputs.new_sphere = 'fs_LR-deformed_to-fsaverage.L.sphere.32k_fs_LR.
↳surf.gii'
>>> metres.inputs.area_metrics = True
>>> metres.inputs.current_area = 'fsaverage5.L.midthickness_va_avg.10k_fsavg_L.
↳shape.gii'
>>> metres.inputs.new_area = 'fs_LR.L.midthickness_va_avg.32k_fs_LR.shape.gii'
>>> metres.cmdline
'wb_command -metric-resample sub-01_task-rest_bold_space-fsaverage5.L.func.gii
↳ fsaverage5_std_sphere.L.10k_fsavg_L.surf.gii fs_LR-deformed_to-fsaverage.L.
↳ sphere.32k_fs_LR.surf.gii ADAP_BARY_AREA fs_LR-deformed_to-fsaverage.L.
↳ sphere.32k_fs_LR.surf.out -area-metrics fsaverage5.L.midthickness_va_avg.
↳ 10k_fsavg_L.shape.gii fs_LR.L.midthickness_va_avg.32k_fs_LR.shape.gii'
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
    The metric file to resample
    argument: ``%s``, position: 0
method: ('ADAP_BARY_AREA' or 'BARYCENTRIC')
    The method name - ADAP_BARY_AREA method is recommended for ordinary
    metric data, because it should use all data while downsampling,
    unlike BARYCENTRIC. If ADAP_BARY_AREA is used, exactly one of
    area_surfs or area_metrics must be specified
    argument: ``%s``, position: 3
new_sphere: (an existing file name)
    A sphere surface that is in register with <current-sphere> and has
    the desired output mesh
    argument: ``%s``, position: 2
current_sphere: (an existing file name)
    A sphere surface with the mesh that the metric is currently on
    argument: ``%s``, position: 1

[Optional]
out_file: (a file name)
    The output metric
    argument: ``%s``, position: 4
valid_roi_out: (a boolean)
    Output the ROI of vertices that got data from valid source vertices
    argument: ``-valid-roi-out``, position: 9
area_metrics: (a boolean)
    Specify vertex area metrics to do area correction based on
    argument: ``-area-metrics``, position: 5
    mutually_exclusive: area_surfs
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
area_surfs: (a boolean)
```

(continues on next page)

(continued from previous page)

```

Specify surfaces to do vertex area correction based on
argument: ``-area-surfs``, position: 5
mutually_exclusive: area_metrics
current_area: (an existing file name)
    A relevant anatomical surface with <current-sphere> mesh OR a metric
    file with vertex areas for <current-sphere> mesh
    argument: ``%s``, position: 6
roi_metric: (an existing file name)
    Input roi on the current mesh used to exclude non-data vertices
    argument: ``-current-roi %s``, position: 8
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
new_area: (an existing file name)
    A relevant anatomical surface with <current-sphere> mesh OR a metric
    file with vertex areas for <current-sphere> mesh
    argument: ``%s``, position: 7
largest: (a boolean)
    Use only the value of the vertex with the largest weight
    argument: ``-largest``, position: 10

```

Outputs:

```

roi_file: (a file name)
    ROI of vertices that got data from valid source vertices
out_file: (an existing file name)
    the output metric

```


81.1 Bru2

[Link to code](#)

Wraps the executable command Bru2.

Uses bru2nii's Bru2 to convert Bruker files

81.1.1 Examples

```
>>> from nipy.interfaces.bru2nii import Bru2
>>> converter = Bru2()
>>> converter.inputs.input_dir = "brukerdir"
>>> converter.cmdline
'Bru2 -o .../data/brukerdir brukerdir'
```

Inputs:

```
[Mandatory]
input_dir: (an existing directory name)
    Input Directory
    argument: ``%s``, position: -1

[Optional]
output_filename: (a unicode string)
    Output filename (".nii" will be appended, or ".nii.gz" if the "-z"
    compress option is selected)
    argument: ``-o %s``
compress: (a boolean)
    gz compress images (".nii.gz").
    argument: ``-z``
append_protocol_name: (a boolean)
    Append protocol name to output filename.
    argument: ``-p``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
force_conversion: (a boolean)
```

(continues on next page)

(continued from previous page)

```
Force conversion of localizers images (multiple slice orientations).
argument: ``-f``
actual_size: (a boolean)
Keep actual size - otherwise x10 scale so animals match human.
argument: ``-a``
environ: (a dictionary with keys which are a 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
```

Outputs:

```
nii_file: (an existing file name)
```

82.1 C3d

[Link to code](#)

Wraps the executable command `c3d`.

Convert3d is a command-line tool for converting 3D (or 4D) images between common file formats. The tool also includes a growing list of commands for image manipulation, such as thresholding and resampling. The tool can also be used to obtain information about image files. More information on Convert3d can be found at: <https://sourceforge.net/p/c3d/git/ci/master/tree/doc/c3d.md>

82.1.1 Example

```
>>> from nipy.interfaces.c3 import C3d
>>> c3 = C3d()
>>> c3.inputs.in_file = "T1.nii"
>>> c3.inputs.pix_type = "short"
>>> c3.inputs.out_file = "T1.img"
>>> c3.cmdline
'c3d T1.nii -type short -o T1.img'
>>> c3.inputs.is_4d = True
>>> c3.inputs.in_file = "epi.nii"
>>> c3.inputs.out_file = "epi.img"
>>> c3.cmdline
'c4d epi.nii -type short -o epi.img'
```

Inputs:

```
[Mandatory]
in_file: (a list of items which are a file name)
         Input file (wildcard and multiple are supported).
         argument: ``%s``, position: 1

[Optional]
shift: (an integer (int or long) or a float)
       Adds the given constant to every voxel.
       argument: ``-shift %s``
```

(continues on next page)

(continued from previous page)

```

resample: (a unicode string)
    Resamples the image, keeping the bounding box the same, but changing
    the number of voxels in the image. The dimensions can be specified
    as a percentage, for example to double the number of voxels in each
    direction. The -interpolation flag affects how sampling is
    performed.
    argument: ``-resample %s``
out_file: (a file name)
    Output file of last image on the stack.
    argument: ``-o %s``, position: -1
    mutually_exclusive: out_files
is_4d: (a boolean, nipy default value: False)
    Changes command to support 4D file operations (default is false).
out_files: (a list of items which are a file name)
    Write all images on the convert3d stack as multiple files. Supports
    both list of output files or a pattern for the output filenames
    (using %d substitution).
    argument: ``-oo %s``, position: -1
    mutually_exclusive: out_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', nipy default value: {})
    Environment variables
scale: (an integer (int or long) or a float)
    Multiplies the intensity of each voxel in the last image on the
    stack by the given factor.
    argument: ``-scale %s``
interp: ('Linear' or 'NearestNeighbor' or 'Cubic' or 'Sinc' or
    'Gaussian')
    Specifies the interpolation used with -resample and other commands.
    Default is Linear.
    argument: ``-interpolation %s``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
smooth: (a unicode string)
    Applies Gaussian smoothing to the image. The parameter vector
    specifies the standard deviation of the Gaussian kernel.
    argument: ``-smooth %s``
multicomp_split: (a boolean, nipy default value: False)
    Enable reading of multi-component images.
    argument: ``-mcr``, position: 0
pix_type: ('float' or 'char' or 'uchar' or 'short' or 'ushort' or
    'int' or 'uint' or 'double')
    Specifies the pixel type for the output image. By default, images
    are written in floating point (float) format
    argument: ``-type %s``

```

Outputs:

```

out_files: (a list of items which are a file name)

```

82.2 C3dAffineTool

[Link to code](#)

Wraps the executable command c3d_affine_tool.

Converts fsl-style Affine registration into ANTS compatible itk format

82.2.1 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:

```
[Optional]
reference_file: (an existing file name)
    argument: ``-ref %s``, position: 1
itk_transform: (a boolean or a file name)
    Export ITK transform.
    argument: ``-oitk %s``, position: 5
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
fsl2ras: (a boolean)
    argument: ``-fsl2ras``, position: 4
transform_file: (an existing file name)
    argument: ``%s``, 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
source_file: (an existing file name)
    argument: ``-src %s``, position: 2
```

Outputs:

```
itk_transform: (an existing file name)
```


83.1 Dcm2nii

[Link to code](#)

Wraps the executable command dcm2nii.

Uses MRICron's dcm2nii to convert dicom files

83.1.1 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)
    argument: ``%s``, position: -1
    mutually_exclusive: source_names
source_names: (a list of items which are an existing file name)
    argument: ``%s``, position: -1
    mutually_exclusive: source_dir

[Optional]
date_in_filename: (a boolean, nipy default value: True)
    Date in filename
    argument: ``-d``
convert_all_pars: (a boolean, nipy default value: True)
    Convert every image in directory
    argument: ``-v``
reorient_and_crop: (a boolean, nipy default value: False)
    Reorient and crop 3D images
```

(continues on next page)

(continued from previous page)

```

        argument: ``-x``
nii_output: (a boolean, nipy default value: True)
    Save as .nii - if no, create .hdr/.img pair
    argument: ``-n``
collapse_folders: (a boolean, nipy default value: True)
    Collapse input folders
    argument: ``-c``
spm_analyze: (a boolean)
    SPM2/Analyze not SPM5/NIfTI
    argument: ``-s``
    mutually_exclusive: nii_output
environ: (a dictionary with keys which are a bytes or None or a value
    of class 'str' and with values which are a bytes or None or a
    value of class 'str', nipy default value: {})
    Environment variables
source_in_filename: (a boolean, nipy default value: False)
    Source filename
    argument: ``-f``
anonymize: (a boolean, nipy default value: True)
    Remove identifying information
    argument: ``-a``
protocol_in_filename: (a boolean, nipy default value: True)
    Protocol in filename
    argument: ``-p``
events_in_filename: (a boolean, nipy default value: True)
    Events (series/acq) in filename
    argument: ``-e``
reorient: (a boolean)
    Reorient image to nearest orthogonal
    argument: ``-r``
gzip_output: (a boolean, nipy default value: False)
    Gzip output (.gz)
    argument: ``-g``
args: (a unicode string)
    Additional parameters to the command
    argument: ``%s``
id_in_filename: (a boolean, nipy default value: False)
    ID in filename
    argument: ``-i``
config_file: (an existing file name)
    Load settings from specified inifile
    argument: ``-b %s``
output_dir: (an existing directory name)
    Output dir - if unspecified, source directory is used
    argument: ``-o %s``

```

Outputs:

```

reoriented_and_cropped_files: (a list of items which are an existing
    file name)
converted_files: (a list of items which are an existing file name)
reoriented_files: (a list of items which are an existing file name)
bvecs: (a list of items which are an existing file name)
bvals: (a list of items which are an existing file name)

```

83.2 Dcm2niix

[Link to code](#)

Wraps the executable command dcm2niix.

Uses Chris Rorden’s dcm2niix to convert dicom files

83.2.1 Examples

```
>>> from nipy.interfaces.dcm2nii import Dcm2niix
>>> converter = Dcm2niix()
>>> converter.inputs.source_dir = 'dicomdir'
>>> converter.inputs.compression = 5
>>> converter.inputs.output_dir = 'ds005'
>>> converter.cmdline
'dcm2niix -b y -z y -5 -x n -t n -m n -o ds005 -s n -v n dicomdir'
>>> converter.run()
```

In the example below, we note that the current version of dcm2niix converts any files in the directory containing the files in the list. We also do not support nested filenames with this option. **Thus all files must have a common root directory.**

```
>>> converter = Dcm2niix()
>>> converter.inputs.source_names = ['functional_1.dcm', 'functional_2.dcm']
>>> converter.inputs.compression = 5
>>> converter.inputs.output_dir = 'ds005'
>>> converter.cmdline
'dcm2niix -b y -z y -5 -x n -t n -m n -o ds005 -s n -v n .'
>>> converter.run()
```

Inputs:

```
[Mandatory]
source_dir: (an existing directory name)
    A directory containing dicom files to be converted
    argument: ``%s``, position: -1
    mutually_exclusive: source_names
source_names: (a list of items which are an existing file name)
    A set of filenames to be converted. Note that the current version
    (1.0.20180328) of dcm2niix converts any files in the directory. To
    only convert specific files they should be in an isolated directory
    argument: ``%s``, position: -1
    mutually_exclusive: source_dir

[Optional]
has_private: (a boolean, nipy default value: False)
    Flag if text notes include private patient details
    argument: ``-t``
comment: (a unicode string)
    Comment stored as NIfTI aux_file
    argument: ``-c %s``
compress: ('y' or 'i' or 'n' or '3', nipy default value: y)
    Gzip compress images - [y=pigz, i=internal, n=no, 3=no,3D]
    argument: ``-z %s``
out_filename: (a unicode string)
    Output filename template (%a=antenna (coil) number, %c=comments,
    %d=description, %e=echo number, %f=folder name, %i=ID of patient,
    %j=seriesInstanceUID, %k=studyInstanceUID, %m=manufacturer, %n=name
```

(continues on next page)

(continued from previous page)

```

of patient, %p=protocol, %s=series number, %t=time, %u=acquisition
number, %v=vendor, %x=study ID; %z=sequence name)
argument: ``-f %s``
series_numbers: (a list of items which are a unicode string)
Selectively convert by series number - can be used up to 16 times
argument: ``-n %s...``
compression: (1 or 2 or 3 or 4 or 5 or 6 or 7 or 8 or 9)
Gz compression level (1=fastest, 9=smallest)
argument: ``-d``
crop: (a boolean, nipy default value: False)
Crop 3D T1 acquisitions
argument: ``-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
output_dir: (an existing directory name, nipy default value: .)
Output directory
argument: ``-o %s``
single_file: (a boolean, nipy default value: False)
Single file mode
argument: ``-s``
philips_float: (a boolean)
Philips precise float (not display) scaling
argument: ``-p``
merge_imgs: (a boolean, nipy default value: False)
merge 2D slices from same series
argument: ``-m``
verbose: (a boolean, nipy default value: False)
Verbose output
argument: ``-v``
ignore_deriv: (a boolean)
Ignore derived, localizer and 2D images
argument: ``-i``
bids_format: (a boolean, nipy default value: True)
Create a BIDS sidecar file
argument: ``-b``
args: (a unicode string)
Additional parameters to the command
argument: ``%s``
anon_bids: (a boolean)
Anonymize BIDS
argument: ``-ba``
requires: bids_format

```

Outputs:

```

bids: (a list of items which are an existing file name)
converted_files: (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)

```

84.1 CopyMeta

[Link to code](#)

Copy meta data from one Nifti file to another. Useful for preserving meta data after some processing steps.

Inputs:

```
[Mandatory]
src_file: (an existing file name)
dest_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)
```

84.2 DcmStack

[Link to code](#)

Create one Nifti file from a set of DICOM files. Can optionally embed meta data.

84.2.1 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]
include_regexes: (a list of items which are any value)
                  Meta data to include, overriding any exclude filters
force_read: (a boolean, nipy default value: True)
             Force reading files without DICM marker
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)
exclude_regexes: (a list of items which are any value)
                  Meta data to exclude, supplementing any default exclude filters
embed_meta: (a boolean)
             Embed DICOM meta data into result
out_path: (a directory name)
           output path, current working directory if not set
```

Outputs:

```
out_file: (an existing file name)
```

84.3 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]
include_regexes: (a list of items which are any value)
                  Meta data to include, overriding any exclude filters
force_read: (a boolean, nipy default value: True)
             Force reading files without DICM marker
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)
exclude_regexes: (a list of items which are any value)
                  Meta data to exclude, supplementing any default exclude filters
embed_meta: (a boolean)
             Embed DICOM meta data into result
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
```

84.4 LookupMeta

[Link to code](#)

Lookup meta data values from a Nifti with embedded meta data.

84.4.1 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]
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
in_file: (an existing file name)
    The input Nifti file
```

Outputs:

None

84.5 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]
sort_order: (a unicode string or a list of items which are any value)
    One or more meta data keys to sort files by.
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)
merge_dim: (an integer (int or long))
    Dimension to merge along. If not specified, the last singular or
    non-existent dimension is used.
out_path: (a directory name)
    output path, current working directory if not set
```

Outputs:

```
out_file: (an existing file name)
          Merged Nifti file
```

84.6 NiftiGeneratorBase

[Link to code](#)

Base class for interfaces that produce Nifti files, potentially with embedded meta data.

Inputs:

```
None
```

Outputs:

```
None
```

84.7 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]
split_dim: (an integer (int or long))
           Dimension to split along. If not specified, the last dimension is
           used.
out_path: (a directory name)
          output path, current working directory if not set
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)
```

Outputs:

```
out_list: (a list of items which are an existing file name)
          Split Nifti files
```

84.8 make_key_func()

[Link to code](#)

84.9 sanitize_path_comp()

[Link to code](#)

85.1 SlicerCommandLine

[Link to code](#)

Wraps the executable command Slicer3.

Experimental Slicer wrapper. Work in progress.

Inputs:

```
[Optional]
environ: (a dictionary with keys which are a 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
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
module: (a unicode string)
        name of the Slicer command line module you want to use
```

Outputs:

None

86.1 Reorient

[Link to code](#)

Conform an image to a given orientation

Flips and reorder the image data array so that the axes match the directions indicated in `orientation`. The default RAS orientation corresponds to the first axis being ordered from left to right, the second axis from posterior to anterior, and the third axis from inferior to superior.

For oblique images, the original orientation is considered to be the closest plumb orientation.

No resampling is performed, and thus the output image is not de-obliques or registered to any other image or template.

The effective transform is calculated from the original affine matrix to the reoriented affine matrix.

86.1.1 Examples

If an image is not reoriented, the original file is not modified

```
>>> import numpy as np
>>> from nipy.interfaces.image import Reorient
>>> reorient = Reorient(orientation='LPS')
>>> reorient.inputs.in_file = 'segmentation0.nii.gz'
>>> res = reorient.run()
>>> res.outputs.out_file
'segmentation0.nii.gz'
```

```
>>> print_affine(np.loadtxt(res.outputs.transform))
1.  0.  0.  0.
0.  1.  0.  0.
0.  0.  1.  0.
0.  0.  0.  1.
```

```
>>> reorient.inputs.orientation = 'RAS'
>>> res = reorient.run()
>>> res.outputs.out_file
'.../segmentation0_ras.nii.gz'
```

```
>>> print_affine(np.loadtxt(res.outputs.transform))
-1.  0.  0.  60.
 0. -1.  0.  72.
 0.  0.  1.   0.
 0.  0.  0.   1.
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        Input image

[Optional]
orientation: ('RAS' or 'RAI' or 'RPS' or 'RPI' or 'LAS' or 'LAI' or
              'LPS' or 'LPI' or 'RSA' or 'RSP' or 'RIA' or 'RIP' or 'LSA' or
              'LSP' or 'LIA' or 'LIP' or 'ARS' or 'ARI' or 'ALS' or 'ALI' or
              'PRS' or 'PRI' or 'PLS' or 'PLI' or 'ASR' or 'ASL' or 'AIR' or
              'AIL' or 'PSR' or 'PSL' or 'PIR' or 'PIL' or 'SRA' or 'SRP' or
              'SLA' or 'SLP' or 'IRA' or 'IRP' or 'ILA' or 'ILP' or 'SAR' or
              'SAL' or 'SPR' or 'SPL' or 'IAR' or 'IAL' or 'IPR' or 'IPL',
              nipy default value: RAS)
        Target axis orientation
```

Outputs:

```
transform: (an existing file name)
           Affine transform from input orientation to output
out_file: (an existing file name)
          Reoriented image
```

86.2 Rescale

[Link to code](#)**Rescale an image**

Rescales the non-zero portion of `in_file` to match the bounds of the non-zero portion of `ref_file`. Reference values in the input and reference images are defined by the `percentile` parameter, and the reference values in each image are identified and the remaining values are scaled accordingly. In the case of `percentile == 0`, the reference values are the maxima and minima of each image. If the `invert` parameter is set, the input file is inverted prior to rescaling.

86.2.1 Examples

To use a high-resolution T1w image as a registration target for a T2* image, it may be useful to invert the T1w image and rescale to the T2* range. Using the 1st and 99th percentiles may reduce the impact of outlier voxels.

```
>>> from nipy.interfaces.image import Rescale
>>> invert_t1w = Rescale(invert=True)
>>> invert_t1w.inputs.in_file = 'structural.nii'
>>> invert_t1w.inputs.ref_file = 'functional.nii'
>>> invert_t1w.inputs.percentile = 1.
>>> res = invert_t1w.run()
```

Inputs:

```
[Mandatory]
ref_file: (an existing file name)
          Skull-stripped reference image
```

(continues on next page)

(continued from previous page)

```
in_file: (an existing file name)
        Skull-stripped image to rescale

[Optional]
invert: (a boolean)
        Invert contrast of rescaled image
percentile: (0.0 <= a floating point number <= 50.0, nipy default
            value: 0.0)
            Percentile to use for reference to allow for outliers - 1 indicates
            the 1st and 99th percentiles in the input file will be mapped to the
            99th and 1st percentiles in the reference; 0 indicates minima and
            maxima will be mapped
```

Outputs:

```
out_file: (an existing file name)
        Rescaled image
```


87.1 BIDSDataGrabber

[Link to code](#)

BIDS datagrabber module that wraps around pybids to allow arbitrary querying of BIDS datasets.

87.1.1 Examples

By default, the BIDSDataGrabber fetches anatomical and functional images from a project, and makes BIDS entities (e.g. subject) available for filtering outputs.

```
>>> bg = BIDSDataGrabber()
>>> bg.inputs.base_dir = 'ds005/'
>>> bg.inputs.subject = '01'
>>> results = bg.run()
```

Dynamically created, user-defined output fields can also be defined to return different types of outputs from the same project. All outputs are filtered on common entities, which can be explicitly defined as infields.

```
>>> bg = BIDSDataGrabber(infields = ['subject'], outfields = ['dwi'])
>>> bg.inputs.base_dir = 'ds005/'
>>> bg.inputs.subject = '01'
>>> bg.inputs.output_query['dwi'] = dict(modality='dwi')
>>> results = bg.run()
```

Inputs:

```
[Mandatory]
base_dir: (an existing directory name)
          Path to BIDS Directory.

[Optional]
return_type: ('file' or 'namedtuple', nipy default value: file)
raise_on_empty: (a boolean, nipy default value: True)
                 Generate exception if list is empty for a given field
strict: (a boolean)
         Return only BIDS "proper" files (e.g., ignore derivatives/,
         sourcedata/, etc.)
```

(continues on next page)

(continued from previous page)

```
output_query: (a dictionary with keys which are a unicode string and
               with values which are a dictionary with keys which are any value
               and with values which are any value)
               Queries for outfield outputs
```

Outputs:

None

87.2 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.

87.2.1 Examples

```
>>> from nipyne.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]
max_depth: (an integer (int or long))
            The maximum depth to search beneath the root_paths
match_regex: (a unicode string, nipyne default value: (.+))
              Regular expression for matching paths.
min_depth: (an integer (int or long))
            The minimum depth to search beneath the root paths
ignore_regexes: (a list of items which are any value)
                 List of regular expressions, if any match the path it will be
```

(continues on next page)

(continued from previous page)

```

        ignored.
unpack_single: (a boolean, nipy default value: False)
    Unpack single results from list

```

Outputs:

None

87.3 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

87.3.1 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]
template: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

        Layout used to get files. relative to base directory if defined
sort_filelist: (a boolean)
        Sort the filelist that matches the template

[Optional]
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
base_directory: (an existing directory name)
                Path to the base directory consisting of subject data.
drop_blank_outputs: (a boolean, nipy default value: False)
                Remove ``None`` entries from output lists
raise_on_empty: (a boolean, nipy default value: True)
                Generate exception if list is empty for a given field

```

Outputs:**None**

87.4 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 comes from 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 regexp_substitutions.

This interface **cannot** be used in a MapNode as the inputs are defined only when the connect statement is executed.

87.4.1 Examples

```

>>> ds = DataSink()
>>> ds.inputs.base_directory = 'results_dir'

```

(continues on next page)

(continued from previous page)

```
>>> 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:

```
[Optional]
_outputs: (a dictionary with keys which are a unicode string and with
          values which are any value, nipype default value: {})
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
local_copy: (a unicode string)
            Copy files locally as well as to S3 bucket
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
encrypt_bucket_keys: (a boolean)
                    Flag indicating whether to use S3 server-side AES-256 encryption
strip_dir: (a directory name)
            path to strip out of filename
remove_dest_dir: (a boolean, nipype default value: False)
                remove dest directory when copying dirs
regexp_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`
parameterization: (a boolean, nipype default value: True)
                store output in parametrized structure
bucket: (any value)
        Boto3 S3 bucket for manual override of bucket
base_directory: (a directory name)
                Path to the base directory for storing data.
container: (a unicode string)
            Folder within base directory in which to store output
```

Outputs:

```
out_file: (any value)
          datasink output
```

87.5 FreeSurferSource

[Link to code](#)

Generates freesurfer subject info from their directories

87.5.1 Examples

```
>>> from nipyne.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: (an existing directory name)
              Freesurfer subjects directory.

[Optional]
hemi: ('both' or 'lh' or 'rh', nipyne default value: both)
      Selects hemisphere specific outputs
```

Outputs:

```
aseg: (an existing file name)
      Volumetric map of regions from automatic segmentation
volume: (a list of items which are an existing file name)
        Surface maps of cortical volume
label: (a list of items which are an existing file name)
        Volume and surface label files
wmparc_stats: (a list of items which are an existing file name)
              White matter parcellation statistics file
wmparc: (an existing file name)
        Aparc parcellation projected into subcortical white matter
nu: (an existing file name)
     Non-uniformity corrected whole-head volume
sphere: (a list of items which are an existing file name)
        Spherical surface meshes
sulc: (a list of items which are an existing file name)
       Surface maps of sulcal depth
aparc_a2009s_stats: (a list of items which are an existing file name)
                    Aparc a2009s parcellation statistics files
T1: (an existing file name)
     Intensity normalized whole-head volume
inflated: (a list of items which are an existing file name)
          Inflated surface meshes
smoothwm: (a list of items which are an existing file name)
           Smoothed original surface meshes
jacobian_white: (a list of items which are an existing file name)
                Distortion required to register to spherical atlas
curv_pial: (a list of items which are an existing file name)
            Curvature of pial surface
sphere_reg: (a list of items which are an existing file name)
             Spherical registration file
aparc_aseg: (a list of items which are an existing file name)
```

(continues on next page)

(continued from previous page)

```

    Aparc parcellation projected into aseg volume
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
curv_stats: (a list of items which are an existing file name)
    Curvature statistics files
aseg_stats: (a list of items which are an existing file name)
    Automated segmentation statistics file
brain: (an existing file name)
    Intensity normalized brain-only volume
pial: (a list of items which are an existing file name)
    Gray matter/pia mater surface meshes
ribbon: (a list of items which are an existing file name)
    Volumetric maps of cortical ribbons
rawavg: (an existing file name)
    Volume formed by averaging input images
avg_curv: (a list of items which are an existing file name)
    Average atlas curvature, sampled to subject
curv: (a list of items which are an existing file name)
    Maps of surface curvature
norm: (an existing file name)
    Normalized skull-stripped volume
white: (a list of items which are an existing file name)
    White/gray matter surface meshes
orig: (an existing file name)
    Base image conformed to Freesurfer space
graymid: (a list of items which are an existing file name)
    Graymid/midthickness surface meshes
entorhinal_exvivo_stats: (a list of items which are an existing file
    name)
    Entorhinal exvivo statistics files
thickness: (a list of items which are an existing file name)
    Surface maps of cortical thickness
BA_stats: (a list of items which are an existing file name)
    Brodmann Area statistics files
annot: (a list of items which are an existing file name)
    Surface annotation files
filled: (an existing file name)
    Subcortical mass volume
aparc_stats: (a list of items which are an existing file name)
    Aparc parcellation statistics files
wm: (an existing file name)
    Segmented white-matter volume
brainmask: (an existing file name)
    Skull-stripped (brain-only) volume

```

87.6 IOBase

[Link to code](#)

Inputs:

None

Outputs:

None

87.7 JSONFileGrabber

[Link to code](#)

Datagrabber interface that loads a json file and generates an output for every first-level object

87.7.1 Example

```
>>> import pprint
>>> from nipyee.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 = os.path.join(datadir, 'jsongrabber.txt')
>>> res = jsonSource.run()
>>> pprint.pprint(res.outputs.get())
{'param1': 'exampleStr', 'param2': 4, 'param3': 1.0}
```

Inputs:

```
[Optional]
in_file: (an existing file name)
        JSON source file
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
```

Outputs:

None

87.8 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:

```
[Optional]
_outputs: (a dictionary with keys which are any value and with values
           which are any value, nipyne default value: {})
out_file: (a file name)
           JSON sink file
in_dict: (a dictionary with keys which are any value and with values
          which are any value, nipyne default value: {})
 JSON dictionary
```

Outputs:

```
out_file: (a file name)
          JSON sink file
```

87.9 MySQLSink

[Link to code](#)

Very simple frontend for storing values into MySQL database.

87.9.1 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
host: (a unicode string, nipyne default value: localhost)
      mutually_exclusive: config
      requires: username, password
database_name: (a unicode string)
               Otherwise known as the schema name
table_name: (a unicode string)

[Optional]
username: (a unicode string)
password: (a unicode string)
```

Outputs:

None

87.10 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
template: (a unicode string)
    Layout used to get files. Relative to bucket_path if defined. Uses
    regex rather than glob style formatting.
sort_filelist: (a boolean)
    Sort the filelist that matches the template

[Optional]
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.
region: (a unicode string, nipyre default value: us-east-1)
    Region of s3 bucket
bucket_path: (a unicode string, nipyre default value: )
    Location within your bucket for subject data.
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
anon: (a boolean, nipyre 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.
raise_on_empty: (a boolean, nipyre default value: True)
    Generate exception if list is empty for a given field
```

Outputs:

None

87.11 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.

87.11.1 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)
```

Outputs:

```
None
```

87.12 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

87.12.1 Examples

```
>>> from nipyne.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.
template: (a unicode string)
    Layout used to get files. relative to base directory if defined
hostname: (a unicode string)
    Server hostname.
sort_filelist: (a boolean)
    Sort the filelist that matches the template

[Optional]
template_expression: ('fnmatch' or 'regex', nipyne default value:
    fnmatch)
    Use either fnmatch or regex to express templates
username: (a unicode string)
    Server username.
password: (a string)
    Server password.
download_files: (a boolean, nipyne default value: True)
    If false it will return the file names without downloading them
raise_on_empty: (a boolean, nipyne 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
drop_blank_outputs: (a boolean, nipyne default value: False)
    Remove ``None`` entries from output lists
ssh_log_to_file: (a unicode string, nipyne default value: )
    If set SSH commands will be logged to the given file
```

Outputs:

None

87.13 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.

87.13.1 Examples

```
>>> import pprint
>>> from nipyne 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()) # doctest:
{'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:

```
[Optional]
force_lists: (a boolean or a list of items which are a unicode
              string, nipyne 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
raise_on_empty: (a boolean, nipyne default value: True)
                Raise an exception if a template pattern matches no files.
base_directory: (an existing directory name)
                Root path common to templates.
sort_filelist: (a boolean, nipyne default value: True)
                When matching mutliple files, return them in sorted order.
```

Outputs:

None

87.14 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]
experiment_id: (a unicode string)
                Set to workflow name
config: (a file name)
                mutually_exclusive: server
project_id: (a unicode string)
                Project in which to store the outputs
subject_id: (a unicode string)
                Set to subject id
server: (a unicode string)
```

(continues on next page)

(continued from previous page)

```

        mutually_exclusive: config
        requires: user, pwd

[Optional]
assessor_id: (a unicode string)
    Option to customize outputs representation in XNAT - assessor level
    will be used with specified id
    mutually_exclusive: reconstruction_id
_outputs: (a dictionary with keys which are a unicode string and with
    values which are any value, nipype default value: {})
share: (a boolean, nipype default value: False)
    Option to share the subjects from the original project instead of
    creating new ones when possible - the created experiments are then
    shared back to the original project
pwd: (a string)
user: (a unicode string)
reconstruction_id: (a unicode string)
    Option to customize outputs representation in XNAT - reconstruction
    level will be used with specified id
    mutually_exclusive: assessor_id
cache_dir: (a directory name)

```

Outputs:

None

87.15 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.

87.15.1 Examples

```
>>> from nipype.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'
```

(continues on next page)

(continued from previous page)

```
>>> 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
server: (a unicode string)
    mutually_exclusive: config
    requires: user, pwd
query_template: (a unicode string)
    Layout used to get files. Relative to base directory if defined

[Optional]
cache_dir: (a directory name)
    Cache directory
pwd: (a string)
user: (a unicode 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, nipy default value: {'outfiles':
    []})
    Information to plug into template
```

Outputs:

```
None
```

87.16 add_traits()

[Link to code](#)

Add traits to a traitled class.

All traits are set to Undefined by default

87.17 copytree()

[Link to code](#)Recursively copy a directory tree using `nipy.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.

87.18 push_file()

[Link to code](#)

87.19 quote_id()

[Link to code](#)

87.20 `unquote_id()`

[Link to code](#)

88.1 MeshFix

[Link to code](#)

Wraps the executable 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.

88.1.1 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)
          argument: ``%s``, position: 1

[Optional]
dont_clean: (a boolean)
            Don't Clean
            argument: ``--no-clean``
join_closest_components: (a boolean)
                         Join the closest pair of components.
                         argument: ``-jc``
```

(continues on next page)

(continued from previous page)

```

    mutually_exclusive: join_closest_components
finetuning_inwards: (a boolean)
    Used to fine-tune the minimal distance between surfaces.
    argument: ``--fineTuneIn``, position: -3
    requires: finetuning_distance, finetuning_substeps
uniform_remeshing_steps: (an integer (int or long))
    Number of steps for uniform remeshing of the whole mesh
    argument: ``-u %d``
    requires: uniform_remeshing_vertices
number_of_biggest_shells: (an integer (int or long))
    Only the N biggest shells are kept
    argument: ``--shells %d``
in_file2: (an existing file name)
    argument: ``%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', nipyte default value: {})
    Environment variables
dilation: (an integer (int or long))
    Dilate the surface by d. d < 0 means shrinking.
    argument: ``--dilate %d``
quiet_mode: (a boolean)
    Quiet mode, don't write much to stdout.
    argument: ``-q``
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.
    argument: ``--decouple-inin %d``
save_as_vrml: (a boolean)
    Result is saved in VRML1.0 format (.wrl)
    argument: ``--wrl``
    mutually_exclusive: save_as_stl, save_as_freesurfer_mesh
join_overlapping_largest_components: (a boolean)
    Join 2 biggest components if they overlap, remove the rest.
    argument: ``-j``
    mutually_exclusive: join_closest_components
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)
    argument: ``%d``, position: -1
    requires: finetuning_distance
out_filename: (a file name)
    The output filename for the fixed mesh file
    argument: ``-o %s``
uniform_remeshing_vertices: (an integer (int or long))
    Constrains the number of vertices. Must be used with
    uniform_remeshing_steps
    argument: ``--vertices %d``
    requires: uniform_remeshing_steps
save_as_freesurfer_mesh: (a boolean)
    Result is saved in freesurfer mesh format
    argument: ``--fsmesh``
    mutually_exclusive: save_as_vrml, save_as_stl
save_as_stl: (a boolean)

```

(continues on next page)

(continued from previous page)

```

    Result is saved in stereolithographic format (.stl)
    argument: ``--stl``
    mutually_exclusive: save_as_vrml, save_as_freesurfer_mesh
remove_handles: (a boolean)
    Remove handles
    argument: ``--remove-handles``
output_type: ('stl' or 'msh' or 'wrl' or 'vrml' or 'fs' or 'off',
    nipy default value: off)
    The output type to save the file as.
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.
    argument: ``--decouple-outin %d``
x_shift: (an integer (int or long))
    Shifts the coordinates of the vertices when saving. Output must be
    in FreeSurfer format
    argument: ``--smooth %d``
laplacian_smoothing_steps: (an integer (int or long))
    The number of laplacian smoothing steps to apply
    argument: ``--smooth %d``
set_intersections_to_one: (a boolean)
    If the mesh contains intersections, return value = 1.If saved in
    gmsh format, intersections will be highlighted.
    argument: ``--intersect``
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.
    argument: ``--decouple-outout %d``
epsilon_angle: (0.0 <= a floating point number <= 2.0)
    Epsilon angle in degrees (must be between 0 and 2)
    argument: ``-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)
    argument: ``%f``, position: -2
    requires: finetuning_substeps
args: (a unicode string)
    Additional parameters to the command
    argument: ``%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.
    argument: ``--cut-inner %d``
finetuning_outwards: (a boolean)
    Similar to finetuning_inwards, but ensures minimal distance in the
    other direction
    argument: ``--fineTuneOut``, position: -3
    mutually_exclusive: finetuning_inwards
    requires: finetuning_distance, finetuning_substeps
cut_outer: (an integer (int or long))
    Remove triangles of 1st that are outside of the 2nd shell.
    argument: ``--cut-outer %d``

```

Outputs:

```
mesh_file: (an existing file name)
           The output mesh file
```

89.1 NilearnBaseInterface

[Link to code](#)

Inputs:

None

Outputs:

None

89.2 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]
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.
in_file: (an existing file name)
    4-D fMRI nii file
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
```

(continues on next page)

(continued from previous page)

```
(background excluded). This list corresponds to the class labels in
label_file in ascending order

[Optional]
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
detrend: (a boolean, nipy default value: False)
    If True, perform detrending using Nilearn.
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.
```

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
```

90.1 PETPVC

[Link to code](#)

Wraps the executable 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)
```

90.1.1 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”

90.1.2 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_y: (a float)
    The full-width at half maximum in mm along y-axis
    argument: ``-y %.4f``
mask_file: (an existing file name)
    Mask image file
    argument: ``-m %s``
in_file: (an existing file name)
    PET image file
    argument: ``-i %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
    argument: ``-p %s``
fwhm_x: (a float)
    The full-width at half maximum in mm along x-axis
    argument: ``-x %.4f``
fwhm_z: (a float)
```

(continues on next page)

(continued from previous page)

```

        The full-width at half maximum in mm along z-axis
        argument: ``-z %.4f``

[Optional]
n_iter: (an integer (int or long), nipy default value: 10)
        Number of iterations
        argument: ``-n %d``
out_file: (a file name)
        Output file
        argument: ``-o %s``
n_deconv: (an integer (int or long), nipy default value: 10)
        Number of deconvolution iterations
        argument: ``-k %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
alpha: (a float, nipy default value: 1.5)
        Alpha value
        argument: ``-a %.4f``
debug: (a boolean, nipy default value: False)
        Prints debug information
        argument: ``-d``
args: (a unicode string)
        Additional parameters to the command
        argument: ``%s``
stop_crit: (a float, nipy default value: 0.01)
        Stopping criterion
        argument: ``-a %.4f``

```

Outputs:

```

out_file: (a file name)
        Output file

```

90.1.3 References:

None

91.1 Quickshear

[Link to code](#)

Wraps the executable command `quickshear`.

Quickshear is a simple geometric defacing algorithm

Given an anatomical image and a reasonable brainmask, Quickshear estimates a shearing plane with the brain mask on one side and the face on the other, zeroing out the face side.

```
>>> from nipy.interfaces.quickshear import Quickshear
>>> qs = Quickshear(in_file='T1.nii', mask_file='brain_mask.nii')
>>> qs.cmdline
'quickshear T1.nii brain_mask.nii T1_defaced.nii'
```

In the absence of a precomputed mask, a simple pipeline can be generated with any tool that generates brain masks:

```
>>> from nipy.pipeline import engine as pe
>>> from nipy.interfaces import utility as niu
>>> from nipy.interfaces.fsl import BET
>>> deface_wf = pe.Workflow('deface_wf')
>>> inputnode = pe.Node(niu.IdentityInterface(['in_file']),
...                      name='inputnode')
>>> outputnode = pe.Node(niu.IdentityInterface(['out_file']),
...                      name='outputnode')
>>> bet = pe.Node(BET(mask=True), name='bet')
>>> quickshear = pe.Node(Quickshear(), name='quickshear')
>>> deface_wf.connect([
...     (inputnode, bet, [('in_file', 'in_file')]),
...     (inputnode, quickshear, [('in_file', 'in_file')]),
...     (bet, quickshear, [('mask_file', 'mask_file')]),
...     (quickshear, outputnode, [('out_file', 'out_file')]),
...     ])
>>> inputnode.inputs.in_file = 'T1.nii'
>>> res = deface_wf.run()
```

Inputs:

```
[Mandatory]
in_file: (an existing file name)
        neuroimage to deface
        argument: ``%s``, position: 1
mask_file: (an existing file name)
        brain mask
        argument: ``%s``, position: 2

[Optional]
buff: (an integer (int or long))
      buffer size (in voxels) between shearing plane and the brain
      argument: ``%d``, position: 4
args: (a unicode string)
      Additional parameters to the command
      argument: ``%s``
environ: (a dictionary with keys which are a bytes or None or a value
         of class 'str' and with values which are a bytes or None or a
         value of class 'str', nipy default value: {})
         Environment variables
out_file: (a file name)
         defaced output image
         argument: ``%s``, position: 3
```

Outputs:

```
out_file: (an existing file name)
         defaced output image
```

91.1.1 References:

None

92.1 `configure_input_data()`

[Link to code](#)

Configure the input data for vtk pipeline object obj. Copied from latest version of mayavi

92.2 `vtk_old()`

[Link to code](#)

Checks if VTK uses the old-style pipeline (VTK<6.0)

92.3 `vtk_output()`

[Link to code](#)

Configure the input data for vtk pipeline object obj.

Developer Guides

- API

Release 1.1.7

Date December 17, 2018, 15:26 PDT

93.1 `caching.memory`

93.1.1 Module: `caching.memory`

Inheritance diagram for `nipytype.caching.memory`:

memory.PipeFunc

memory.Memory

Using nipyte with persistence and lazy recomputation but without explicit name-steps pipeline: getting back scope in command-line based programming.

93.1.2 Classes

Memory

class `nipyte.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

<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.

__init__ (*base_dir*)

Initialize self. See `help(type(self))` for accurate signature.

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

<code>__call__</code> (**kwargs)	Call self as a function.
<code>__init__</code> (<i>interface, base_dir, callback=None</i>)	
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.	

93.1.3 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

93.2 conftest

93.2.1 Module: `conftest`

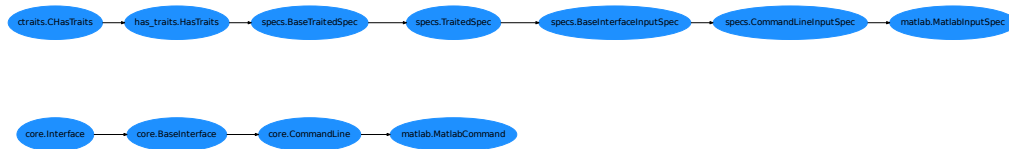
93.2.2 Functions

```
nipyype.conftest.add_np(doctest_namespace)
nipyype.conftest.pytest_unconfigure(config)
```

93.3 interfaces.matlab

93.3.1 Module: `interfaces.matlab`

Inheritance diagram for `nipyype.interfaces.matlab`:



General matlab interface code

93.3.2 Classes

`MatlabCommand`

```
class nipyype.interfaces.matlab.MatlabCommand(matlab_cmd=None, **inputs)
```

Bases: `nipyype.interfaces.base.core.CommandLine`

Interface that runs matlab code

```
>>> import nipyype.interfaces.matlab as matlab
>>> mlab = matlab.MatlabCommand(mfile=False) # don't write script file
>>> mlab.inputs.script = "which('who')"
>>> out = mlab.run()
```

Attributes

always_run Should the interface be always run even if the inputs were not changed?

Only applies to interfaces being run within a workflow context.

can_resume Defines if the interface can reuse partial results after interruption.

cmd sets base command, immutable

cmdline *command* plus any arguments (*args*)

output_spec

terminal_output**version** interfaces should implement a version property**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 <i>MatlabInputSpec</i>
<code>load_inputs_from_json(json_file[, overwrite])</code>	A convenient way to load pre-set inputs from a JSON file.
<code>run([cwd, ignore_exception])</code>	Execute this interface.
<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.

raise_exception	
version_from_command	

__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

Should the interface be always run even if the inputs were not changed? Only applies to interfaces being run within a workflow context.

can_resume

Defines if the interface can reuse partial results after interruption. Only applies to interfaces being run within a workflow context.

cmd

sets base command, immutable

cmdline*command* plus any arguments (*args*) validates arguments and generates command line**classmethod help** (*returnhelp=False*)

Prints class help

input_specalias 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_** = []**resource_monitor = True****run** (*cwd=None, ignore_exception=None, **inputs*)

Execute this interface.

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

Parameters

cwd [specify a folder where the interface should be run]

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.

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>.terminal_output.

terminal_output

version

interfaces should implement a version property

version_from_command (*flag='-v', cmd=None*)

MatlabInputSpec

class nipyte.interfaces.matlab.**MatlabInputSpec** (***kwargs*)

Bases: nipyte.interfaces.base.specs.CommandLineInputSpec

Basic expected inputs to Matlab interface

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>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.

Continued on next page

Table 4 – continued from previous page

<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 trait attributes associated with the current class, that should be GUI visible
<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 interface
<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>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.

Continued on next page

Table 4 – continued from previous page

<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(**kwargs)</code>	Returns traitled class as a dict
<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 trait attributes associated with the current object, that should be GUI visible

add_trait_listener	
class_trait_view	
remove_trait_listener	

__init__ (***kwargs*)

Initialize handlers and inputs

classmethod 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().

classmethod 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.

classmethod class_default_traits_view ()

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

classmethod class_editable_traits ()

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

classmethod 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.

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

classmethod 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.

classmethod 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.

classmethod 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

list_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.

package_version = <Version('1.1.7')>

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.

classmethod 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 (***kwargs*)

Returns traitled class as a dict

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

trait_items_event (*event_trait, name, items_event*)

classmethod 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.

classmethod `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 = {'extended': <class 'traits.trait_notifiers.ExtendedTraitChangeNotifyW

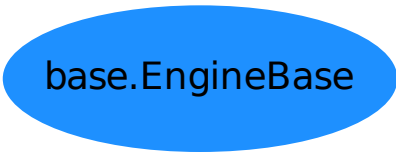
93.3.3 Function

`nipyype.interfaces.matlab.get_matlab_command()`

93.4 pipeline.engine.base

93.4.1 Module: pipeline.engine.base

Inheritance diagram for `nipyype.pipeline.engine.base`:



faces

Defines functionality for pipelined execution of inter-

The *EngineBase* class implements the more general view of a task.

93.4.2 EngineBase

class nipytype.pipeline.engine.base.**EngineBase** (*name=None, base_dir=None*)

Bases: object

Defines common attributes and functions for workflows and nodes.

- Attributes**
- fullname**
 - inputs**
 - internname* Name for expanded iterable
 - name**
 - outputs**

Methods

<i>clone</i> (name)	Clone an EngineBase object
---------------------	----------------------------

load	
save	

__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

internname

 Name for expanded iterable

load (*filename*)

name

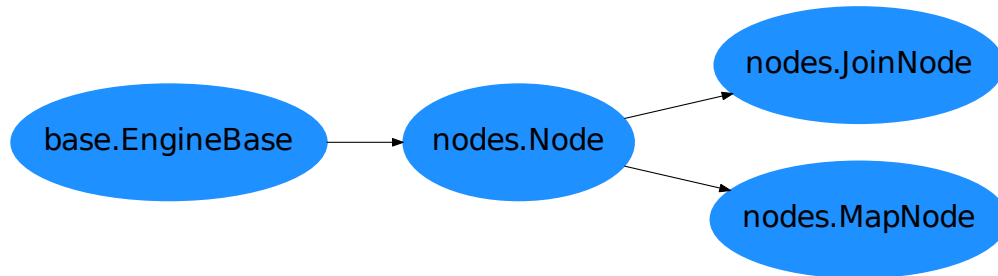
outputs

save (*filename=None*)

93.5 pipeline.engine.nodes

93.5.1 Module: pipeline.engine.nodes

Inheritance diagram for `nipyre.pipeline.engine.nodes`:



Defines functionality for pipelined execution of interfaces
The *Node* class provides core functionality for batch processing.

93.5.2 Classes

JoinNode

class `nipyre.pipeline.engine.nodes.JoinNode`(*interface*, *name*, *joinsource*, *joinfield*=None, *unique*=False, ***kwargs*)

Bases: `nipyre.pipeline.engine.nodes.Node`

Wraps interface objects that join inputs into a list.

Examples

```

>>> import nipyre.pipeline.engine as pe
>>> from nipyre import Node, JoinNode, Workflow
>>> from nipyre.interfaces.utility import IdentityInterface
>>> from nipyre.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')
  
```

Attributes
fullname

inputs The JoinNode inputs include the join field overrides.
interface Return the underlying interface object
itername Name for expanded iterable
joinsource
mem_gb Get estimated memory (GB)
n_procs Get the estimated number of processes/threads
name
needed_outputs
outputs Return the output fields of the underlying interface
result Get result from result file (do not hold it in memory)

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>	Decorate the new <i>is_cached</i> method with hash updating to maintain backwards compatibility.
<code>help()</code>	Print interface help
<code>is_cached([rm_outdated])</code>	Check if the interface has been run previously, and whether cached results are up-to-date.
<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>set_input(parameter, val)</code>	Set interface input value
<code>update(**opts)</code>	Update inputs

load	
save	

__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*)

Decorate the new *is_cached* method with hash updating to maintain backwards compatibility.

help ()

Print interface help

inputs

The JoinNode inputs include the join field overrides.

interface

Return the underlying interface object

is_cached (*rm_outdated=False*)
Check if the interface has been run previously, and whether cached results are up-to-date.

itername
Name for expanded iterable

joinfield = None
the fields to join

joinsource
the join predecessor iterable node

load (*filename*)

mem_gb
Get estimated memory (GB)

n_procs
Get the estimated number of processes/threads

name

needed_outputs

output_dir ()
Return the location of the output directory for the node

outputs
Return the output fields of the underlying interface

result
Get result from result file (do not hold it in memory)

run (*updatehash=False*)
Execute the node in its directory.

Parameters

updatehash: boolean When the hash stored in the output directory as a result of a previous run does not match that calculated for this execution, updatehash=True only updates the hash without re-running.

save (*filename=None*)

set_input (*parameter, val*)
Set interface input value

update (***opts*)
Update inputs

MapNode

class `nipy.pipeline.engine.nodes.MapNode` (*interface, iterfield, name, serial=False, nested=False, **kwargs*)

Bases: `nipy.pipeline.engine.nodes.Node`

Wraps interface objects that need to be iterated on a list of inputs.

Examples

```
>>> from nipy import MapNode
>>> from nipy.interfaces import fsl
>>> realign = MapNode(fsl.MCFLIRT(), 'in_file', 'realign')
>>> realign.inputs.in_file = ['functional.nii',
...                           'functional2.nii',
...                           'functional3.nii']
>>> realign.run()
```

Attributes

fullname

inputs Return the inputs of the underlying interface

interface Return the underlying interface object

itername Name for expanded iterable

mem_gb Get estimated memory (GB)
n_procs Get the estimated number of processes/threads
name
needed_outputs
outputs Return the output fields of the underlying interface
result Get result from result file (do not hold it in memory)

Methods

<code>clone(name)</code>	Clone an EngineBase object
<code>get_output(parameter)</code>	Retrieve a particular output of the node
<code>get_subnodes()</code>	Generate subnodes of a mapnode and write pre-execution report
<code>hash_exists([updatehash])</code>	Decorate the new <i>is_cached</i> method with hash updating to maintain backwards compatibility.
<code>help()</code>	Print interface help
<code>is_cached([rm_outdated])</code>	Check if the interface has been run previously, and whether cached results are up-to-date.
<code>num_subnodes()</code>	Get the number of subnodes to iterate in this MapNode
<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>set_input(parameter, val)</code>	Set interface input value or nodewrapper attribute Priority goes to interface.
<code>update(**opts)</code>	Update inputs

load	
save	

__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 ()

Generate subnodes of a mapnode and write pre-execution report

hash_exists (*updatehash=False*)

Decorate the new `is_cached` method with hash updating to maintain backwards compatibility.

```
help()
    Print interface help
inputs
    Return the inputs of the underlying interface
interface
    Return the underlying interface object
is_cached(rm_outdated=False)
    Check if the interface has been run previously, and whether cached results are up-to-date.
itername
    Name for expanded iterable
load(filename)
mem_gb
    Get estimated memory (GB)
n_procs
    Get the estimated number of processes/threads
name
needed_outputs
num_subnodes()
    Get the number of subnodes to iterate in this MapNode
output_dir()
    Return the location of the output directory for the node
outputs
    Return the output fields of the underlying interface
result
    Get result from result file (do not hold it in memory)
run(updatehash=False)
    Execute the node in its directory.
    Parameters
        updatehash: boolean When the hash stored in the output directory as a result of a previous run does not match that calculated for this execution, updatehash=True only updates the hash without re-running.
save(filename=None)
set_input(parameter, val)
    Set interface input value or nodewrapper attribute Priority goes to interface.
update(**opts)
    Update inputs
```

Node

```
class nipytype.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=None, mem_gb=0.2, **kwargs)
```

Bases: `nipytype.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()
```

Attributes

fullname

inputs Return the inputs of the underlying interface

interface Return the underlying interface object

itername Name for expanded iterable

mem_gb Get estimated memory (GB)

n_procs Get the estimated number of processes/threads

name

needed_outputs

outputs Return the output fields of the underlying interface

result Get result from result file (do not hold it in memory)

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>	Decorate the new <code>is_cached</code> method with hash updating to maintain backwards compatibility.
<code>help()</code>	Print interface help
<code>is_cached([rm_outdated])</code>	Check if the interface has been run previously, and whether cached results are up-to-date.
<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>set_input(parameter, val)</code>	Set interface input value
<code>update(**opts)</code>	Update inputs

load	
save	

__init__ (*interface*, *name*, *iterables=None*, *itersource=None*, *synchronize=False*, *overwrite=None*, *needed_outputs=None*, *run_without_submitting=False*, *n_procs=None*, *mem_gb=0.2*, ***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.itorsource = ('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)]]
```

itorsource: 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*)

Decorate the new *is_cached* method with hash updating to maintain backwards compatibility.

help ()

Print interface help

inputs

Return the inputs of the underlying interface

interface

Return the underlying interface object

is_cached (*rm_outdated=False*)

Check if the interface has been run previously, and whether cached results are up-to-date.

itername

Name for expanded iterable

load (*filename*)

mem_gb

Get estimated memory (GB)

n_procs

Get the estimated number of processes/threads

name

needed_outputs

output_dir ()

Return the location of the output directory for the node

outputs
Return the output fields of the underlying interface

result
Get result from result file (do not hold it in memory)

run (*updatehash=False*)
Execute the node in its directory.

Parameters

updatehash: boolean When the hash stored in the output directory as a result of a previous run does not match that calculated for this execution, updatehash=True only updates the hash without re-running.

save (*filename=None*)

set_input (*parameter, val*)
Set interface input value

update (***opts*)
Update inputs

93.6 pipeline.engine.utils

93.6.1 Module: pipeline.engine.utils

Utility routines for workflow graphs

93.6.2 Functions

`nipyne.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

`nipyne.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.

`nipyne.pipeline.engine.utils.evaluate_connect_function` (*function_source, args, first_arg*)

`nipyne.pipeline.engine.utils.expand_iterables` (*iterables, synchronize=False*)

`nipyne.pipeline.engine.utils.export_graph` (*graph_in, base_dir=None, show=False, use_execgraph=False, show_connectioninfo=False, dotfilename='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 from 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.load_resultfile(path, name)`

Load InterfaceResult file from path

Returns

result [InterfaceResult structure]

aggregate [boolean indicating whether node should aggregate_outputs]

attribute error [boolean indicating whether there was some mismatch in] versions of traits used to store result and hence node needs to rerun

`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, strs

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.nodelist_runner(nodes, updatehash=False, stop_first=False)`

A generator that iterates and over a list of nodes and executes them.

`nipyype.pipeline.engine.utils.save_hashfile(hashfile, hashed_inputs)`

Store a hashfile

`nipyype.pipeline.engine.utils.save_resultfile(result, cwd, name)`

Save a result `pklz` file to `cwd`

`nipyte.pipeline.engine.utils.strip_temp(files, wd)`
 Remove temp from a list of file paths

`nipyte.pipeline.engine.utils.synchronize_iterables(iterables)`
 Synchronize the given iterables in item-wise order.
 Return: the {field: value} dictionary list

Examples

```
>>> from nipyte.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
```

`nipyte.pipeline.engine.utils.topological_sort(graph, depth_first=False)`
 Returns a depth first sorted order if depth_first is True

`nipyte.pipeline.engine.utils.walk(children, level=0, path=None, username=True)`
 Generate all the full paths in a tree, as a dict.

Examples

```
>>> from nipyte.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]
```

`nipyte.pipeline.engine.utils.walk_files(cwd)`

`nipyte.pipeline.engine.utils.walk_outputs(object)`
 Extract every file and directory from a python structure

`nipyte.pipeline.engine.utils.write_report(node, report_type=None, is_mapnode=False)`
 Write a report file for a node

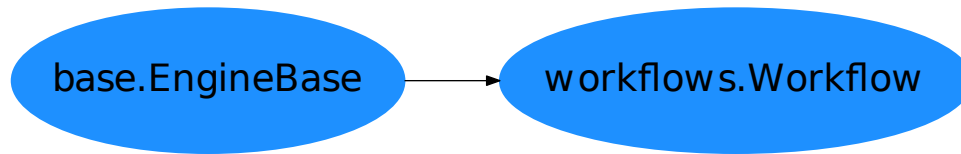
`nipyte.pipeline.engine.utils.write_workflow_prov(graph, filename=None, format='all')`
 Write W3C PROV Model JSON file

`nipyte.pipeline.engine.utils.write_workflow_resources(graph, filename=None, append=None)`
 Generate a JSON file with profiling traces that can be loaded in a pandas DataFrame or processed with JavaScript like D3.js

93.7 pipeline.engine.workflows

93.7.1 Module: pipeline.engine.workflows

Inheritance diagram for `nipyte.pipeline.engine.workflows`:



Defines functionality for pipelined execution of interfaces
 The *Workflow* class provides core functionality for batch processing.

93.7.2 Workflow

class nipyype.pipeline.engine.workflows.**Workflow**(*name*, *base_dir=None*)

Bases: *nipyype.pipeline.engine.base.EngineBase*

Controls the setup and execution of a pipeline of processes.

Attributes

fullname
inputs
internname Name for expanded iterable
name
outputs

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>remove_nodes</i> (nodes)	Remove nodes from a workflow
<i>run</i> ([plugin, plugin_args, updatehash])	Execute the workflow
<i>write_graph</i> ([dotfilename, graph2use, ...])	Generates a graphviz dot file and a png file

load	
save	
write_hierarchical_dotfile	

__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

Name for expanded iterable

list_node_names ()

List names of all nodes in a workflow

load (*filename*)
name
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*)

93.8 sphinxext.plot_workflow

93.8.1 Module: sphinxext.plot_workflow

Inheritance diagram for `nipytype.sphinxext.plot_workflow`:



plot_workflow.ImageFile



plot_workflow.GraphError

nipytype.sphinxext.plot_workflow – Workflow plotting extension

A directive for including a nipytype 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 nipytype.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:

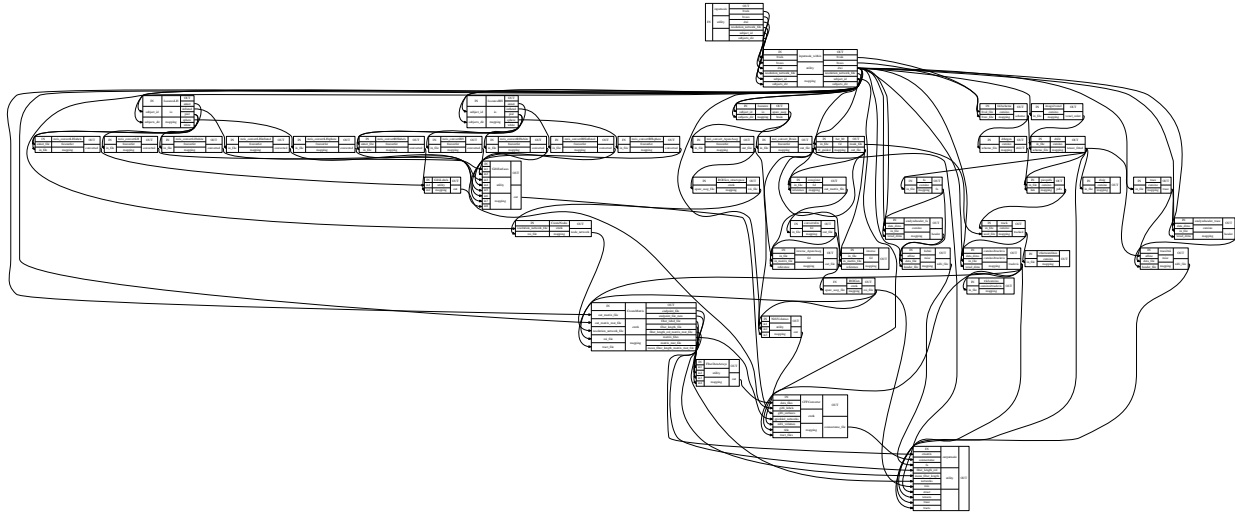
graph2use [{‘hierarchical’, ‘colored’, ‘flat’, ‘orig’, ‘exec’}] Specify the type of graph to be generated.

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.

93.8.2 Classes

GraphError

class nipyype.sphinxext.plot_workflow.**GraphError**

Bases: RuntimeError

Attributes

args

Methods

<i>with_traceback</i>	Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.	–	set
-----------------------	--	---	-----

__init__ (\$self, /, *args, **kwargs)

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 nipyype.sphinxext.plot_workflow.**ImageFile** (basename, dirname)

Bases: object

Methods

filename	
filenames	

__init__ (basename, dirname)

Initialize self. See help(type(self)) for accurate signature.

filename (fmt)

filenames ()

93.8.3 Functions

nipyype.sphinxext.plot_workflow.**contains_doctest** (text)

nipyype.sphinxext.plot_workflow.**get_wf_formats** (config)

nipyype.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.

nipyype.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.

nipyype.sphinxext.plot_workflow.**remove_coding** (text)

Remove the coding comment, which exec doesn’t like.

nipyype.sphinxext.plot_workflow.**render_figures** (code, code_path, output_dir, output_base, context, function_name, config, graph2use, simple_form, context_reset=False, close_figs=False)

Run a nipyype workflow creation script and save the graph in *output_dir*. Save the images under *output_dir* with file names derived from *output_base*

nipyype.sphinxext.plot_workflow.**run** (arguments, content, options, state_machine, state, lineno)

`nipytype.sphinxext.plot_workflow.run_code`(*code*, *code_path*, *ns=None*, *function_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*)

- Developer Guide

Release 1.1.7

Date December 17, 2018, 15:26 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](#)

94.1 Interface Specifications

94.1.1 Before you start

Nipype is maintained by an enthusiastic group of developers, and we're excited to have you join us! In case of trouble, we encourage you to post on [NeuroStars](#) with the *nipype* tag. NeuroStars.org is a platform similar to StackOverflow but dedicated to neuroinformatics. You can also post on the nipype developers mailing list: <http://mail.python.org/mailman/listinfo/neuroimaging>. As we are sharing a mailing list with the nipy community, please add [nipype] to the message title. Alternatively, you're welcome to chat with us in the Nipype [Gitter](#) channel or in the BrainHack [Slack](#) channel. (Click [here](#) to join the Slack workspace.)

94.1.2 Overview

We're using the [Traits](#) (formerly known as 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 it's 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 4.x which can be installed from [pypi](#)

More documentation

Not everything is documented in the User Manual, in those cases the [API docs](#) is your next place to look.

94.1.3 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. 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.FSLTraitSpec`. `FSLTraitSpec` 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):

`TraitSpec`: 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.

94.1.4 Controlling outputs to terminal

It is very likely that the software wrapped within the interface writes to the standard output or the standard error of the terminal. Interfaces provide a means to access and retrieve these outputs, by using the `terminal_output` attribute:

```
import nipy.interfaces.fsl as fsl
mybet = fsl.BET(from_file='bet-settings.json')
mybet.terminal_output = 'file_split'
```

In the example, the `terminal_output = 'file_split'` will redirect the standard output and the standard error to split files (called `stdout.nipype` and `stderr.nipype` respectively). The possible values for `terminal_output` are:

- file** Redirects both standard output and standard error to the same file called `output.nipype`. Messages from both streams will be overlapped as they arrive to the file.
- file_split** Redirects the output streams separately, to `stdout.nipype` and `stderr.nipype` respectively, as described in the example.
- file_stdout** Only the standard output will be redirected to `stdout.nipype` and the standard error will be discarded.
- file_stderr** Only the standard error will be redirected to `stderr.nipype` and the standard output will be discarded.
- stream** Both output streams are redirected to the current logger printing their messages interleaved and immediately to the terminal.
- allatonce** Both output streams will be forwarded to a buffer and stored separately in the `runtime` object that the `run()` method returns. No files are written nor streams printed out to terminal.
- none** Both outputs are discarded

In all cases, except for the 'none' setting of `terminal_output`, the `run()` method will return a “runtime” object that will contain the streams in the corresponding properties (`runtime.stdout` for the standard output, `runtime.stderr` for the standard error, and `runtime.merged` for both when streams are mixed, eg. when using the *file* option).

```
import nipy.interfaces.fsl as fsl
mybet = fsl.BET(from_file='bet-settings.json')
mybet.terminal_output = 'file_split'
```

(continues on next page)

(continued from previous page)

```
...
result = mybet.run()
result.runtime.stdout
' ... captured standard output ...'
```

94.1.5 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 Nipype 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 nipype 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)
```

(continues on next page)

(continued from previous page)

```

...     y = traits.Int
...
>>> foo = Foo()
>>> foo.x
12
>>> foo.y
0

```

Nipype 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 nipype.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',

```

(continues on next page)

(continued from previous page)

```
        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 Nipyne 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 nipyne 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.

94.1.6 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 Nipype start Xvfb before running the interface and redirect X output to it. This is useful for
- `_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 asciii 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(TraitSpec):
    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 `out_file` 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*.

94.1.7 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).

94.1.8 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)
```

(continues on next page)

(continued from previous page)

```

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/nipy/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`.

94.2 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 Nipype 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.

94.2.1 Defining inputs and outputs

In Nipype 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 Nipype) 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 Nipyre 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.

94.2.2 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 `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 nipy.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()
```

94.2.3 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(TraitedSpec):
    inverse_warp = File(exists=True,
                        desc=('Name of output file, containing warps that '
                              'are the "reverse" of those in --warp.'))
```

94.3 How to wrap a MATLAB script

94.3.1 Example 1

This is a minimal script for wrapping MATLAB code. You should replace the MATLAB code template, and define appropriate inputs and outputs.

```
from nipyne.interfaces.matlab import MatlabCommand
from nipyne.interfaces.base import TraitedSpec, \
    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(TraitedSpec):
    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)
```

(continues on next page)

(continued from previous page)

```

# 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 source code

You can download the source code of this example.

94.3.2 Example 2

By subclassing `nipyne.interfaces.matlab.MatlabCommand` for your main class, and `nipyne.interfaces.matlab.MatlabInputSpec` for your input and output spec, you gain access to some useful MATLAB hooks

```

from nipyne.interfaces.base import traits
from nipyne.interfaces.base import TraitedSpec
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(TraitedSpec):
    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()

```

(continues on next page)

(continued from previous page)

```

>>> 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

```

Example source code

You can download the source code of this example.

94.4 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, TraitSpec
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(TraitSpec):
    thresholded_volume = File(exists=True, desc="thresholded volume")

```

(continues on next page)

(continued from previous page)

```

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

```

94.5 Working with *nipyre* source code

Contents:

94.5.1 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](#).

94.5.2 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

94.5.3 Following the latest source

These are the instructions if you just want to follow the latest *nipyre* 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/nipyre.git
```

You now have a copy of the code tree in the new *nipyre* directory.

Updating the code

From time to time you may want to pull down the latest code. Do this with:

```
cd nipyre
git pull
```

The tree in *nipyre* will now have the latest changes from the initial repository.

94.5.4 Making a patch

You've discovered a bug or something else you want to change in *nipyre* .. — 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/nipyre.git
# make a branch for your patching
cd nipyre
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'
```

(continues on next page)

(continued from previous page)

```
# make the patch files
git format-patch -M -C master
```

Then, send the generated patch files to the [nipyre 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/nipyre.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*.

94.5.5 Git for development

Contents:

Making your own copy (fork) of nipyre

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 *nipyre* project, and to suggest some default names.

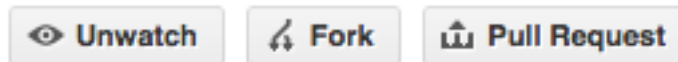
Set up and configure a github account

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 nipyre

1. Log into your *github* account.
2. Go to the *nipyre* github home at *nipyre 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 *nipyre*.

Set up your fork

First you follow the instructions for *Making your own copy (fork) of nipyre*.

Overview

```
git clone git@github.com:your-user-name/nipyre.git
cd nipyre
git remote add upstream git://github.com/nipy/nipyre.git
```

In detail

Clone your fork

1. Clone your fork to the local computer with `git clone git@github.com:your-user-name/nipyre.git`
2. Investigate. Change directory to your new repo: `cd nipyre`. 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 [nipyre github](#) repository, so you can merge in changes from trunk.

Linking your repository to the upstream repo

```
cd nipyre
git remote add upstream git://github.com/nipy/nipyre.git
```

`upstream` here is just the arbitrary name we're using to refer to the main [nipyre](#) repository at [nipyre 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/nipyre.git (fetch)
upstream      git://github.com/nipy/nipyre.git (push)
origin        git@github.com:your-user-name/nipyre.git (fetch)
origin        git@github.com:your-user-name/nipyre.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
  wdiff = diff --color-words

[core]
  editor = vim
```

(continues on next page)

(continued from previous page)

```
[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
```

(continues on next page)

(continued from previous page)

```
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 [nipyte](#) repository, by following *Making your own copy (fork) of nipyte*, *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 [nipyte](#) development repo. Your `master` then will follow the main [nipyte](#) 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 [nipyte](#). 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).

- 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:

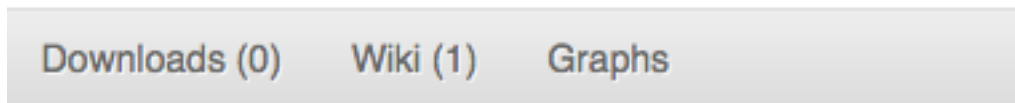
```
git diff master...topic
```

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:

- Go to the URL of your forked repo, say `http://github.com/your-user-name/nipyre.git`.
- 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
```

(continues on next page)

¹ Thanks to Yarik Halchenko for this explanation.

(continued from previous page)

```
git fetch upstream
# merge from upstream
git merge upstream/master
```

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 nipyte into your account, as from *Making your own copy (fork) of nipyte*.

Then, go to your forked repository github page, say `http://github.com/your-user-name/nipyte`. 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/nipyte.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.

94.5.6 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`

94.6 Architecture (discussions from 2009)

This section reflects notes and discussion between developers during the start of the nipyre project in 2009.

94.6.1 Design Guidelines

These are guidelines that the core nipyre 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.

94.6.2 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.

94.6.3 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.

94.6.4 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'
```

(continues on next page)

(continued from previous page)

```
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'))
```

(continues on next page)

(continued from previous page)

```
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'
```

94.7 W3C PROV support

94.7.1 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()
```

94.8 Software using Nipype

94.8.1 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.

94.8.2 BRAINSTools

BRAINSTools is a suite of tools for medical image processing focused on brain analysis.

94.8.3 Brain Imaging Pipelines (BIPs)

BIPs is a set of predefined Nipyne workflows coupled with a graphical interface and ability to save and share workflow configurations. It provides both Nipyne Workflows and Interfaces.

94.8.4 BROCCOLI

BROCCOLI is a piece of software for fast fMRI analysis on many core CPUs and GPUs. It provides Nipyne Interfaces.

94.8.5 Forward

Forward is set of tools simplifying the preparation of accurate electromagnetic head models for EEG forward modeling. It uses Nipyne Workflows and Interfaces.

94.8.6 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 Nipyne Interfaces and Workflows.

94.8.7 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 Nipyne Workflows and Interfaces.

94.8.8 Medimsight

Medimsight is a commercial service medical imaging cloud platform. It uses Nipyne to interface with various neuroimaging software.

94.8.9 MIA

MIA MIA is a a toolkit for gray scale medical image analysis. It provides Nipyne interfaces for easy integration with other software.

94.8.10 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 Nipyne pipeline framework.

94.8.11 OpenfMRI

OpenfMRI is a repository for task based fMRI datasets. It uses Nipyne for automated analysis of the deposited data.

94.8.12 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 Nipyne to process the data.

94.8.13 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 Nipyype Workflows and Interfaces.

94.8.14 tract_querier

tract_querier is a White Matter Query Language tool. It provides Nipyype interfaces.

94.9 Testing nipyype

In order to ensure the stability of each release of Nipyype, 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: 

94.9.1 Installation for developers

To check out the latest development version:

```
git clone https://github.com/nipy/nipyype.git
```

After cloning:

```
cd nipyype
pip install -r requirements.txt
pip install -e .[dev]
```

94.9.2 Test implementation

Nipyype testing framework is built upon **pytest**.

After installation in developer mode, the tests can be run with the following command at the root folder of the project

```
pytest -v --doctest-modules nipyype
```

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 with a local copy of MATLAB installed, 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).

Skipped tests

Nipyype 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 skipped tests.

A few tests in Nipype 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. Note, that the test execution time can increase significantly with these additional tests.

Xfailed tests

Some tests are expect to fail until the code will be changed or for other reasons.

94.9.3 Testing Nipype using Docker

Nipype is tested inside Docker containers and users can use nipype images to test local versions. First, install the [Docker Engine](#). Nipype has one base docker image called `nipype/nipype:base`, that contains several useful tools

(FreeSurfer, AFNI, FSL, ANTs, etc.), and additional test images for specific Python versions: `py27` for Python 2.7 and `py36` for Python 3.6. Users can pull the nipype image for Python 3.6 as follows:

```
docker pull nipype/nipype:py36
```

In order to test a local version of nipype you can run test within container as follows:

```
docker run -it -v $PWD:/src/nipype --rm nipype/nipype:py36 py.test -v --doctest-  
↪modules /src/nipype/nipype
```

94.9.4 Additional comments

If the project is tested both on your local OS and within a Docker container, you might have to remove all `__pycache__` directories before switching between your OS and a container.

Bibliography

- [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 Nalcioglu 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, 376
nipype.algorithms.icc, 379
nipype.algorithms.misc, 402
nipype.algorithms.modelgen, 409
nipype.caching.memory, 1552
nipype.interfaces.base.specs, 613
nipype.interfaces.base.support, 613
nipype.interfaces.base.traits_extension, 613
nipype.interfaces.brainsuite.brainsuite, 643
nipype.interfaces.cmtk.cmtk, 712
nipype.interfaces.cmtk.nbs, 716
nipype.interfaces.cmtk.nx, 718
nipype.interfaces.cmtk.parcellation, 720
nipype.interfaces.dcmstack, 1514
nipype.interfaces.dipy.preprocess, 733
nipype.interfaces.freesurfer.utils, 902
nipype.interfaces.fsl.model, 984
nipype.interfaces.io, 1535
nipype.interfaces.matlab, 1554
nipype.interfaces.minc.base, 1049
nipype.interfaces.minc.testdata, 1095
nipype.interfaces.mrtrix.convert, 1127
nipype.interfaces.mrtrix.tensors, 1146
nipype.interfaces.niftyreg.base, 1201
nipype.interfaces.vtkbase, 1549
nipype.pipeline.engine.base, 1565
nipype.pipeline.engine.nodes, 1566
nipype.pipeline.engine.utils, 1573
nipype.pipeline.engine.workflows, 1576
nipype.workflows.data, 3
nipype.workflows.dmri.camino.connectivity_mapping, 5
nipype.workflows.dmri.camino.diffusion, 6
nipype.workflows.dmri.camino.group_connectivity, 7
nipype.workflows.dmri.connectivity.group_connectivity, 8
nipype.workflows.dmri.connectivity.nx, 11
nipype.workflows.dmri.dipy.denoise, 13
nipype.workflows.dmri.dtitk.tensor_registration, 14
nipype.workflows.dmri.fsl.artifacts, 16
nipype.workflows.dmri.fsl.dti, 38
nipype.workflows.dmri.fsl.epi, 42
nipype.workflows.dmri.fsl.tbss, 55
nipype.workflows.dmri.fsl.utils, 69
nipype.workflows.dmri.mrtrix.connectivity_mapping, 76
nipype.workflows.dmri.mrtrix.diffusion, 77
nipype.workflows.dmri.mrtrix.group_connectivity, 79
nipype.workflows.fmri.fsl.estimate, 81
nipype.workflows.fmri.fsl.preprocess, 86
nipype.workflows.fmri.spm.preprocess, 98
nipype.workflows.misc.utils, 105
nipype.workflows.rsfmri.fsl.resting, 107
nipype.workflows.smri.ants.ANTSBuildTemplate, 113
nipype.workflows.smri.ants.antsRegistrationBuildTemplate, 115
nipype.workflows.smri.freesurfer.autorecon1, 116
nipype.workflows.smri.freesurfer.autorecon2, 116
nipype.workflows.smri.freesurfer.bem, 116
nipype.workflows.smri.freesurfer.recon, 118
nipype.workflows.smri.freesurfer.utils, 120
nipype.workflows.smri.niftyreg.groupwise,

Symbols

- `__init__()` (nipyype.caching.memory.Memory method), 1552
 - `__init__()` (nipyype.caching.memory.PipeFunc method), 1553
 - `__init__()` (nipyype.interfaces.matlab.MatlabCommand method), 1555
 - `__init__()` (nipyype.interfaces.matlab.MatlabInputSpec method), 1558
 - `__init__()` (nipyype.pipeline.engine.base.EngineBase method), 1565
 - `__init__()` (nipyype.pipeline.engine.nodes.JoinNode method), 1567
 - `__init__()` (nipyype.pipeline.engine.nodes.MapNode method), 1569
 - `__init__()` (nipyype.pipeline.engine.nodes.Node method), 1571
 - `__init__()` (nipyype.pipeline.engine.workflows.Workflow method), 1576
 - `__init__()` (nipyype.sphinxext.plot_workflow.GraphError method), 1581
 - `__init__()` (nipyype.sphinxext.plot_workflow.ImageFile method), 1581
- ## A
- ABoverlap, 501
 - ACompCor, 369
 - ACPCTransform, 1424
 - ActivationCount, 417
 - `add_class_trait()` (nipyype.interfaces.matlab.MatlabInputSpec class method), 1558
 - `add_nodes()` (nipyype.pipeline.engine.workflows.Workflow method), 1576
 - `add_np()` (in module nipyype.confest), 1554
 - `add_trait()` (nipyype.interfaces.matlab.MatlabInputSpec method), 1558
 - `add_trait_category()` (nipyype.interfaces.matlab.MatlabInputSpec class method), 1558
 - `add_trait_listener()` (nipyype.interfaces.matlab.MatlabInputSpec method), 1559
 - AddCSVColumn, 391
 - AddCSVRow, 391
 - AddNoise, 392
 - AddScalarVolumes, 1379
 - AddXFormToHeader, 854
 - Affine, 746
 - AffineInitializer, 599
 - AffineRegistration, 1402
 - AffineTask, 747
 - AffScalarVol, 743
 - `affScalarVolTask`, 754
 - `AffSymTensor3DVol`, 744
 - `affSymTensor3DVolTask`, 755
 - AFNIPythonCommand, 419
 - AFNItoNIFTI, 502
 - `aggregate_outputs()` (nipyype.interfaces.matlab.MatlabCommand method), 1555
 - AlignEpiAnatPy, 429
 - `all_trait_names()` (nipyype.interfaces.matlab.MatlabInputSpec method), 1559
 - Allineate, 431
 - `always_run` (nipyype.interfaces.matlab.MatlabCommand attribute), 1555
 - Analyze2nii, 1482
 - AnalyzeHeader, 651
 - AnalyzeWarp, 769
 - antsIntroduction, 558
 - AntsJointFusion, 582
 - Aparc2Aseg, 855
 - Apas2Aseg, 857
 - APMQball, 731
 - ApplyDeformations, 1460
 - ApplyInverseDeformation, 1482
 - ApplyMask, 857, 942
 - ApplyTOPUP, 925
 - ApplyTransform, 1483
 - ApplyTransforms, 575
 - ApplyTransformsToPoints, 577
 - ApplyVolTransform, 808

ApplyWarp, 770, 986
 ApplyXFM, 988
 AR1Image, 942
 args (nipy.sphinxext.plot_workflow.GraphError attribute), 1581
 ArtifactDetect, 413
 AssertEqual, 1489
 Atropos, 584
 Autobox, 503
 Automask, 439
 AutoTcorrelate, 438
 AutoTLRC, 436
 Average, 1049
 AverageAffineTransform, 600
 AverageImages, 601
 AverageNetworks, 716
 AvScale, 1014
 Axialize, 504

B

B0Calc, 985
 Bandpass, 440
 base_trait() (nipy.interfaces.matlab.MatlabInputSpec method), 1559
 BaseInterface, 609
 BBox, 1052
 BBRegister, 810
 BDP, 615
 Beast, 1053
 BEDPOSTX5, 905
 BestLinReg, 1055
 BET, 991
 Bfc, 623
 BIDSDa  aGrabber, 1521
 BigAverage, 1056
 Binarize, 778
 BinaryMaskEditorBasedOnLandmarks, 1345
 BinaryMaths, 943, 1224
 BinaryMathsInteger, 1226
 BinaryStats, 1234
 BinThresh, 758
 BinThreshTask, 759
 Blob, 1057
 Blur, 1058
 BlurInMask, 442
 BlurToFWHM, 443
 BrainExtraction, 586
 BrainMask, 1173
 BRAINSABC, 1333
 BRAINSAlignMSP, 1347
 BRAINSClipInferior, 1348
 BRAINSConstellationDetector, 1336
 BRAINSConstellationModeler, 1349
 BRAINSCreateLabelMapFromProbabilityMaps, 1340

BRAINSCut, 1341
 BRAINSDemonWarp, 1325, 1425
 BRAINSEyeDetector, 1351
 BRAINSFit, 1315, 1416
 BRAINSInitializedControlPoints, 1351
 BRAINSLandmarkInitializer, 1352
 BRAINSLinearModelerEPCA, 1353
 BRAINSLmkTransform, 1354
 BRAINSMultiSTAPLE, 1343
 BRAINSMush, 1354
 BRAINSPosteriorToContinuousClass, 1247
 BRAINSResample, 1323, 1423
 BRAINSResize, 1325
 BRAINSROIAuto, 1344, 1434
 BRAINSSnapShotWriter, 1356
 BRAINSTalairach, 1248
 BRAINSTalairachMask, 1249
 BRAINSTransformConvert, 1357
 BRAINSTransformFromFiducials, 1329
 BRAINSTrimForegroundInDirection, 1358
 BrickStat, 506
 Bru2, 1501
 Bse, 625
 BSplineDeformableRegistration, 1404
 BSplineToDeformationField, 1398
 Bucket, 507
 BuildConnectome, 1159
 buildtemplateparallel, 560

C

C3d, 1503
 C3dAffineTool, 1504
 cache() (nipy.caching.memory.Memory method), 1552
 CALabel, 812
 Calc, 508, 1060
 CalcCoregAffine, 1484
 CalcTopNCC, 1219
 CalculateMedian, 393
 CalculateNormalizedMoments, 393
 Camino2Trackvis, 705
 can_resume (nipy.interfaces.matlab.MatlabCommand attribute), 1555
 CannyEdge, 1302
 CannySegmentationLevelSetImageFilter, 1302
 CANormalize, 814
 CRegister, 815
 CastScalarVolume, 1380
 Cat, 510
 CatMatvec, 511
 CenterMass, 513
 Cerebro, 627
 CFFBaseInterface, 709
 CFFConverter, 713
 ChangeDataType, 944

- CheckerBoardFilter, 1383
 - CheckTalairachAlignment, 858
 - class_default_traits_view()
 - (nipyype.interfaces.matlab.MatlabInputSpec
 - class method), 1559
 - class_editable_traits() (nipyype.interfaces.matlab.MatlabInputSpec
 - class method), 1559
 - class_trait_names() (nipyype.interfaces.matlab.MatlabInputSpec
 - class method), 1559
 - class_trait_view() (nipyype.interfaces.matlab.MatlabInputSpec
 - class method), 1559
 - class_trait_view_elements()
 - (nipyype.interfaces.matlab.MatlabInputSpec
 - class method), 1559
 - class_traits() (nipyype.interfaces.matlab.MatlabInputSpec
 - class method), 1559
 - class_visible_traits() (nipyype.interfaces.matlab.MatlabInputSpec
 - class method), 1559
 - Classifier, 939
 - clean_working_directory()
 - (in module
 - nipyype.pipeline.engine.utils), 1573
 - Cleaner, 939
 - CleanUpOverlapLabels, 1359
 - clear_previous_runs() (nipyype.caching.memory.Memory
 - method), 1553
 - clear_runs_since()
 - (nipyype.caching.memory.Memory
 - method), 1553
 - ClipLevel, 444
 - clone()
 - (nipyype.pipeline.engine.base.EngineBase
 - method), 1565
 - clone() (nipyype.pipeline.engine.nodes.JoinNode method), 1567
 - clone()
 - (nipyype.pipeline.engine.nodes.MapNode
 - method), 1569
 - clone() (nipyype.pipeline.engine.nodes.Node method), 1572
 - clone()
 - (nipyype.pipeline.engine.workflows.Workflow
 - method), 1577
 - clone_traits() (nipyype.interfaces.matlab.MatlabInputSpec
 - method), 1559
 - Cluster, 961
 - cmd
 - (nipyype.interfaces.matlab.MatlabCommand at-
 - tribute), 1555
 - cmdline
 - (nipyype.interfaces.matlab.MatlabCommand at-
 - tribute), 1555
 - CoherenceAnalyzer, 1245
 - CommandLine, 609
 - CommandLineDtk, 743
 - compareTractInclusion, 1264
 - CompCor, 370
 - Complex, 1015
 - ComposeMultiTransform, 602
 - ComposeXfm, 747
 - ComposeXfmTask, 748
 - ComputeDVARs, 371
 - ComputeEigensystem, 662
 - ComputeFractionalAnisotropy, 663
 - ComputeMask, 1241
 - ComputeMeanDiffusivity, 664
 - ComputeMeshWarp, 381
 - ComputeTDI, 1174
 - ComputeTensorTrace, 665
 - Concatenate, 780
 - ConcatenateLTA, 816
 - configure_traits() (nipyype.interfaces.matlab.MatlabInputSpec
 - method), 1559
 - Conmat, 649
 - connect()
 - (nipyype.pipeline.engine.workflows.Workflow
 - method), 1577
 - ConstrainedSphericalDeconvolution, 1139
 - contains_doctest()
 - (in module
 - nipyype.sphinxext.plot_workflow), 1581
 - Contrast, 859
 - ContrastMgr, 963
 - Convert, 1063
 - ConvertDset, 514
 - ConvertScalarImageToRGB, 606
 - ConvertWarp, 1017
 - ConvertXFM, 1019
 - Copy, 515, 1064
 - copy_traits() (nipyype.interfaces.matlab.MatlabInputSpec
 - method), 1560
 - copyable_trait_names() (nipyype.interfaces.matlab.MatlabInputSpec
 - method), 1560
 - CopyGeom, 1020
 - CopyMeta, 1511
 - Coregister, 1460
 - Cortex, 629
 - CorticalThickness, 588
 - count_iterables()
 - (in module
 - nipyype.pipeline.engine.utils), 1573
 - CreateJacobianDeterminantImage, 603
 - CreateMatrix, 709
 - CreateNifti, 394
 - CreateNodes, 711
 - CreateTiledMosaic, 607
 - CreateWarped, 1462
 - CSD, 734
 - CSVReader, 1493
 - Curvature, 861
 - CurvatureAnisotropicDiffusion, 1384
 - CurvatureStats, 862
- ## D
- DARTEL, 1462
 - DARTELENorm2MNI, 1464
 - DataFinder, 1522
 - DataGrabber, 1523

- DataSink, 1524
 - Dcm2nii, 1507
 - Dcm2niix, 1508
 - DcmStack, 1511
 - Deconvolve, 419
 - default_traits_view() (nipy.interfaces.matlab.MatlabInputSpec.
method), 1560
 - DegreeCentrality, 445
 - Denoise, 732
 - DenoiseImage, 591
 - Despike, 447
 - Detrend, 448
 - Dewisp, 630
 - Dfs, 631
 - DICOMConvert, 817
 - DicomImport, 1485
 - DicomToNrrdConverter, 1367
 - Diffeo, 749
 - DiffeoScalarVol, 750
 - diffeoScalarVolTask, 756
 - DiffeoSymTensor3DVOL, 751
 - diffeoSymTensor3DVOLTask, 757
 - DiffeoTask, 752
 - DiffusionTensorScalarMeasurements, 1374
 - DiffusionTensorStreamlineTrack, 1147
 - DiffusionWeightedVolumeMasking, 1375
 - DilateImage, 945, 1303
 - DilateMask, 1304
 - DipyBaseInterface, 732
 - DipyDiffusionInterface, 732
 - Directions2Amplitude, 1142
 - disconnect() (nipy.pipeline.engine.workflows.Workflow
method), 1577
 - Distance, 385, 394
 - DistanceMap, 909
 - DistanceMaps, 1305
 - Dot, 516
 - DT2NIFTI, 653
 - DTI, 738
 - dtiaverage, 1257
 - dtiestim, 1258
 - DTIexport, 1369
 - DTIFit, 666, 907
 - DTIimport, 1370
 - dtiprocess, 1260
 - DTIRecon, 721
 - DTITracker, 722
 - DTLUTGen, 667
 - DTMetric, 669
 - DualRegression, 964
 - Dump, 1065
 - DumpBinaryTrainingVectors, 1305
 - DWI2SphericalHarmonicsImage, 1141
 - DWI2Tensor, 1128
 - DWICompare, 1254
 - DWIConvert, 1255
 - DWIDenoise, 1163
 - DWIExtract, 1177
 - DWIJointRicianLMMSEFilter, 1371
 - DWIRicianLMMSEFilter, 1372
 - DWISimpleCompare, 1254
 - DWIToDTIEstimation, 1373
 - DwiTool, 1190
 - DWIUnbiasedNonLocalMeansFilter, 1399
- ## E
- ECM, 448
 - Eddy, 928
 - EddyCorrect, 931
 - Edge3, 518
 - edit_traits() (nipy.interfaces.matlab.MatlabInputSpec
method), 1560
 - editable_traits() (nipy.interfaces.matlab.MatlabInputSpec
method), 1561
 - EditTransform, 773
 - EditWMwithAseg, 818
 - EM, 1217
 - EMRegister, 846
 - EMSegmentCommandLine, 1435
 - EMSegmentTransformToNewFormat, 1445
 - EngineBase (class in nipy.pipeline.engine.base), 1565
 - EPIDeWarp, 926
 - EpiReg, 932
 - Erode, 1129
 - ErodeImage, 946, 1306
 - ErrorMap, 385
 - ESLR, 1346
 - EstimateContrast, 1239, 1447
 - EstimateFOD, 1166
 - EstimateModel, 1448
 - EstimateResponseForSH, 1143
 - EstimateResponseSH, 735
 - EulerNumber, 863
 - Eval, 519
 - evaluate_connect_function() (in module
nipy.pipeline.engine.utils), 1573
 - expand_iterables() (in module
nipy.pipeline.engine.utils), 1573
 - ExpertAutomatedRegistration, 1405
 - export() (nipy.pipeline.engine.workflows.Workflow
method), 1577
 - export_graph() (in module nipy.pipeline.engine.utils),
1573
 - Extract, 1066
 - ExtractMainComponent, 864
 - extractNrrdVectorIndex, 1265
 - ExtractROI, 1021
 - ExtractSkeleton, 1388

F

FactorialDesign, 1450
 FAST, 994
 fcsv_to_hdf5, 1364
 FEAT, 966
 FEATModel, 966
 FEATRegister, 967
 FeatureExtractor, 940
 fiberprocess, 1293
 fiberstats, 1292
 fibertrack, 1295
 FiducialRegistration, 1429
 FieldMap, 1465
 filename() (nipy.sphinxext.plot_workflow.ImageFile
 method), 1581
 filenames() (nipy.sphinxext.plot_workflow.ImageFile
 method), 1581
 FillLesions, 1222
 FILMGLS, 967
 FilterRegressor, 1022
 FilterTracks, 1149
 Fim, 450
 FindCenterOfBrain, 1359
 FindShPeaks, 1144
 FindTheBiggest, 910
 FIRST, 996
 FitAsl, 1187
 FitDwi, 1192
 FitGLM, 1240
 FitMSPParams, 819
 FitQt1, 1197
 FitTensor, 1168
 FixTopology, 864
 FLAMEO, 970
 FlippedDifference, 1307
 FLIRT, 997
 FNIRT, 1001
 format_dot() (in module nipy.pipeline.engine.utils),
 1573
 format_node() (in module nipy.pipeline.engine.utils),
 1574
 Fourier, 451
 FramewiseDisplacement, 373
 FreeSurferSource, 1525
 FUGUE, 1005
 fullname (nipy.pipeline.engine.base.EngineBase
 attribute), 1565
 fullname (nipy.pipeline.engine.nodes.JoinNode at-
 tribute), 1567
 fullname (nipy.pipeline.engine.nodes.MapNode
 attribute), 1569
 fullname (nipy.pipeline.engine.nodes.Node attribute),
 1572

fullname (nipy.pipeline.engine.workflows.Workflow
 attribute), 1577
 Function, 1494
 FuseSegmentations, 775
 FuzzyOverlap, 386, 395
 FWHMx, 521

G

GaussianBlurImageFilter, 1385
 GCOR, 523
 Generate5tt, 1178
 generate_expanded_graph() (in module
 nipy.pipeline.engine.utils), 1574
 GenerateBrainClippedImage, 1307
 GenerateCsfClippedFromClassifiedImage, 1300
 GenerateDirections, 1145
 GenerateEdgeMapImage, 1251
 GenerateLabelMapFromProbabilityMap, 1361
 GeneratePurePlugMask, 1252
 GenerateSummedGradientImage, 1308
 GenerateTestImage, 1309
 GenerateWhiteMatterMask, 1130
 Gennlxfm, 1069
 GenWarpFields, 557
 get() (nipy.interfaces.matlab.MatlabInputSpec
 method), 1561
 get_all_files() (in module nipy.pipeline.engine.utils),
 1574
 get_hashval() (nipy.interfaces.matlab.MatlabInputSpec
 method), 1561
 get_levels() (in module nipy.pipeline.engine.utils),
 1574
 get_matlab_command() (in module
 nipy.interfaces.matlab), 1564
 get_node() (nipy.pipeline.engine.workflows.Workflow
 method), 1577
 get_output() (nipy.pipeline.engine.nodes.JoinNode
 method), 1567
 get_output() (nipy.pipeline.engine.nodes.MapNode
 method), 1569
 get_output() (nipy.pipeline.engine.nodes.Node
 method), 1572
 get_print_name() (in module
 nipy.pipeline.engine.utils), 1574
 get_subnodes() (nipy.pipeline.engine.nodes.MapNode
 method), 1569
 get_traitsfree() (nipy.interfaces.matlab.MatlabInputSpec
 method), 1561
 get_wf_formats() (in module
 nipy.sphinxext.plot_workflow), 1581
 GLM, 972
 GLMFit, 782
 GradientAnisotropicDiffusion, 1386
 GradientAnisotropicDiffusionImageFilter, 1309

[GraphError](#) (class in [nipype.sphinxext.plot_workflow](#)),
[1581](#)
[GrayscaleFillHoleImageFilter](#), [1391](#)
[GrayscaleGrindPeakImageFilter](#), [1391](#)
[GrayscaleModelMaker](#), [1438](#)
[GroupAndStack](#), [1512](#)
[gtractAnisotropyMap](#), [1266](#)
[gtractAverageBvalues](#), [1266](#)
[gtractClipAnisotropy](#), [1267](#)
[gtractConcatDwi](#), [1271](#)
[gtractCopyImageOrientation](#), [1272](#)
[gtractCoRegAnatomy](#), [1268](#)
[gtractCoregBvalues](#), [1272](#)
[gtractCostFastMarching](#), [1274](#)
[gtractCreateGuideFiber](#), [1276](#)
[gtractFastMarchingTracking](#), [1276](#)
[gtractFiberTracking](#), [1278](#)
[gtractImageConformity](#), [1281](#)
[gtractInvertBSplineTransform](#), [1282](#)
[gtractInvertDisplacementField](#), [1282](#)
[gtractInvertRigidTransform](#), [1283](#)
[gtractResampleAnisotropy](#), [1284](#)
[gtractResampleB0](#), [1285](#)
[gtractResampleCodeImage](#), [1286](#)
[gtractResampleDWIInPlace](#), [1287](#)
[gtractResampleFibers](#), [1288](#)
[gtractTensor](#), [1289](#)
[gtractTransformToDisplacementField](#), [1291](#)
[Gunzip](#), [395](#)

H

[HammerAttributeCreator](#), [1310](#)
[HARDIMat](#), [724](#)
[has_metadata\(\)](#) ([nipype.interfaces.matlab.MatlabInputSpec](#)
[method](#)), [1561](#)
[has_traits_interface\(\)](#) ([nipype.interfaces.matlab.MatlabInputSpec](#)
[method](#)), [1561](#)
[hash_exists\(\)](#) ([nipype.pipeline.engine.nodes.JoinNode](#)
[method](#)), [1567](#)
[hash_exists\(\)](#) ([nipype.pipeline.engine.nodes.MapNode](#)
[method](#)), [1569](#)
[hash_exists\(\)](#) ([nipype.pipeline.engine.nodes.Node](#)
[method](#)), [1572](#)
[help\(\)](#) ([nipype.interfaces.matlab.MatlabCommand](#) class
[method](#)), [1555](#)
[help\(\)](#) ([nipype.pipeline.engine.nodes.JoinNode](#) method),
[1567](#)
[help\(\)](#) ([nipype.pipeline.engine.nodes.MapNode](#) method),
[1570](#)
[help\(\)](#) ([nipype.pipeline.engine.nodes.Node](#) method), [1572](#)
[Hemisplit](#), [632](#)
[Hist](#), [452](#)
[HistogramMatching](#), [1389](#)
[HistogramMatchingFilter](#), [1252](#)

I

[ICA_AROMA](#), [903](#)
[ICC](#), [379](#)
[IdentityInterface](#), [1489](#)
[Image2Voxel](#), [654](#)
[ImageFile](#) (class in [nipype.sphinxext.plot_workflow](#)),
[1581](#)
[ImageLabelCombine](#), [1390](#)
[ImageMaths](#), [1023](#)
[ImageMeants](#), [1024](#)
[ImageRegionPlotter](#), [1361](#)
[ImageStats](#), [703](#), [1025](#)
[input_spec](#) ([nipype.interfaces.matlab.MatlabCommand](#)
[attribute](#)), [1555](#)
[inputs](#) ([nipype.pipeline.engine.base.EngineBase](#) at-
[tribute](#)), [1565](#)
[inputs](#) ([nipype.pipeline.engine.nodes.JoinNode](#) attribute),
[1567](#)
[inputs](#) ([nipype.pipeline.engine.nodes.MapNode](#) attribute),
[1570](#)
[inputs](#) ([nipype.pipeline.engine.nodes.Node](#) attribute),
[1572](#)
[inputs](#) ([nipype.pipeline.engine.workflows.Workflow](#) at-
[tribute](#)), [1577](#)
[insertMidACPCpoint](#), [1364](#)
[IntensityDifferenceMetric](#), [1414](#)
[interface](#) ([nipype.pipeline.engine.nodes.JoinNode](#) at-
[tribute](#)), [1567](#)
[interface](#) ([nipype.pipeline.engine.nodes.MapNode](#) at-
[tribute](#)), [1570](#)
[interface](#) ([nipype.pipeline.engine.nodes.Node](#) attribute),
[1572](#)
[InvWarp](#), [1026](#)
[IOBase](#), [1527](#)
[is_cached\(\)](#) ([nipype.pipeline.engine.nodes.JoinNode](#)
[method](#)), [1567](#)
[is_cached\(\)](#) ([nipype.pipeline.engine.nodes.MapNode](#)
[method](#)), [1570](#)
[is_cached\(\)](#) ([nipype.pipeline.engine.nodes.Node](#)
[method](#)), [1572](#)
[IsotropicSmooth](#), [947](#)
[items\(\)](#) ([nipype.interfaces.matlab.MatlabInputSpec](#)
[method](#)), [1561](#)
[itername](#) ([nipype.pipeline.engine.base.EngineBase](#)
[attribute](#)), [1565](#)
[itername](#) ([nipype.pipeline.engine.nodes.JoinNode](#) at-
[tribute](#)), [1568](#)
[itername](#) ([nipype.pipeline.engine.nodes.MapNode](#) at-
[tribute](#)), [1570](#)
[itername](#) ([nipype.pipeline.engine.nodes.Node](#) attribute),
[1572](#)
[itername](#) ([nipype.pipeline.engine.workflows.Workflow](#)
[attribute](#)), [1577](#)

J

Jacobian, 866
 JistBrainMgdmSegmentation, 1097
 JistBrainMp2rageDuraEstimation, 1099
 JistBrainMp2rageSkullStripping, 1100
 JistBrainPartialVolumeFilter, 1101
 JistCortexSurfaceMeshInflation, 1102
 JistIntensityMp2rageMasking, 1103
 JistLaminarProfileCalculator, 1104
 JistLaminarProfileGeometry, 1105
 JistLaminarProfileSampling, 1106
 JistLaminarROI Averaging, 1107
 JistLaminarVolumetricLayering, 1108
 joinfield (nipy.pipeline.engine.nodes.JoinNode attribute), 1568
 JoinNode (class in nipy.pipeline.engine.nodes), 1566
 joinsource (nipy.pipeline.engine.nodes.JoinNode attribute), 1568
 JointFusion, 592
 JointHistogram, 1362
 JSONFileGrabber, 1528
 JSONFileSink, 1528

K

KellyKapowski, 594

L

L2Model, 974
 Label2Annot, 786
 Label2Label, 787
 Label2Vol, 789
 LabelConfig, 1161
 LabelConvert, 1162
 LabelFusion, 1220
 LabelGeometry, 604
 LabelMapSmoothing, 1439
 landmarksConstellationAligner, 1365
 landmarksConstellationWeights, 1366
 LaplacianThickness, 596
 Level1Design, 975, 1451
 LFCD, 453
 LibraryBaseInterface, 610
 LinearRegistration, 1408
 LinRecon, 696
 list_node_names() (nipy.pipeline.engine.workflows.Workflow method), 1577
 load() (nipy.pipeline.engine.base.EngineBase method), 1565
 load() (nipy.pipeline.engine.nodes.JoinNode method), 1568
 load() (nipy.pipeline.engine.nodes.MapNode method), 1570
 load() (nipy.pipeline.engine.nodes.Node method), 1572
 load() (nipy.pipeline.engine.workflows.Workflow method), 1577
 load_inputs_from_json() (nipy.interfaces.matlab.MatlabCommand method), 1555
 load_resultfile() (in module nipy.pipeline.engine.utils), 1574
 LocalBistat, 524
 Localstat, 526
 LookupMeta, 1512
 LTACConvert, 866

M

MakeAverageSubject, 878
 MakeDyadicVectors, 911
 MakeSurfaces, 879
 MapNode (class in nipy.pipeline.engine.nodes), 1568
 mark_wf_labels() (in module nipy.sphinxext.plot_workflow), 1581
 Maskave, 455
 MaskScalarVolume, 1381
 MaskTool, 529
 Math, 1070
 MathsCommand, 948, 1227
 Matlab2CSV, 396
 MatlabCommand (class in nipy.interfaces.matlab), 1554
 MatlabInputSpec (class in nipy.interfaces.matlab), 1556
 maxcurvature, 1291
 MaxImage, 949
 MaxnImage, 950
 MCFLIRT, 1008
 MeanImage, 951
 Means, 455
 MeasureImageSimilarity, 562
 MedianFilter3D, 1135
 MedianImage, 952
 MedianImageFilter, 1387
 MedicAlgorithmImageCalculator, 1110
 MedicAlgorithmLesionToads, 1111
 MedicAlgorithmMipavReorient, 1113
 MedicAlgorithmN3, 1115
 MedicAlgorithmSPECTRE2010, 1116
 MedicAlgorithmThresholdToBinaryMask, 1120
 MELODIC, 976
 mem_gb (nipy.pipeline.engine.nodes.JoinNode attribute), 1568
 mem_gb (nipy.pipeline.engine.nodes.MapNode attribute), 1570
 mem_gb (nipy.pipeline.engine.nodes.Node attribute), 1572
 Memory (class in nipy.caching.memory), 1552
 Merge, 530, 1028, 1228, 1490

merge_bundles() (in module nipype.pipeline.engine.utils), 1574

merge_dict() (in module nipype.pipeline.engine.utils), 1574

MergeCNetworks, 714

MergeCSVFiles, 396

MergeModels, 1440

MergeNifti, 1513

MergeROIs, 397

MESD, 697

Mesh2PVE, 1182

MeshFix, 1537

MeshWarpMaths, 382

MetricResample, 1497

MinImage, 953

MNIBiasCorrection, 820

ModelFit, 670

ModelMaker, 1441

ModelToLabelMap, 1444

modify_paths() (in module nipype.pipeline.engine.utils), 1574

ModifyAffine, 398

MotionOutliers, 1029

MpiCommandLine, 610

MPRtoMNI305, 847

MRConvert, 1131, 1179

MRICConvert, 822

MRICoreg, 848

MRIFill, 868

MRIMarchingCubes, 869

MRIPretess, 870

MRIsCALabel, 827

MRIsCalc, 872

MRIsCombine, 873

MRIsConvert, 874

MRIsExpand, 876

MRIsInflate, 877

MRISPreproc, 790

MRISPreprocReconAll, 792

MRITessellate, 871

MRMath, 1181

MRMultiply, 1132

MRTransform, 1133

MRTrix2TrackVis, 1127

MRTrix3Base, 1159

MRTrixViewer, 1134

MS_LDA, 794

MultiImageMaths, 954

MultipleRegressDesign, 979

MultipleRegressionDesign, 1452

MultiplyImages, 605

MultiplyScalarVolumes, 1382

MultiResolutionAffineRegistration, 1409

MySQLSink, 1529

N

N4BiasFieldCorrection, 597

N4ITKBiasFieldCorrection, 1392

n_procs (nipype.pipeline.engine.nodes.JoinNode attribute), 1568

n_procs (nipype.pipeline.engine.nodes.MapNode attribute), 1570

n_procs (nipype.pipeline.engine.nodes.Node attribute), 1572

name (nipype.pipeline.engine.base.EngineBase attribute), 1565

name (nipype.pipeline.engine.nodes.JoinNode attribute), 1568

name (nipype.pipeline.engine.nodes.MapNode attribute), 1570

name (nipype.pipeline.engine.nodes.Node attribute), 1572

name (nipype.pipeline.engine.workflows.Workflow attribute), 1578

needed_outputs (nipype.pipeline.engine.nodes.JoinNode attribute), 1568

needed_outputs (nipype.pipeline.engine.nodes.MapNode attribute), 1570

needed_outputs (nipype.pipeline.engine.nodes.Node attribute), 1572

NeighborhoodMean, 1311

NeighborhoodMedian, 1312

NetworkBasedStatistic, 715

NetworkXMetrics, 717

NewSegment, 1466

NIFTIDT2Camino, 655

NiftiGeneratorBase, 1514

NiftyFitCommand, 1190

NiftyRegCommand, 1201

NiftySegCommand, 1217

NilearnBaseInterface, 1541

NipyBaseInterface, 1239

nipype.algorithms.confounds (module), 376

nipype.algorithms.icc (module), 379

nipype.algorithms.misc (module), 402

nipype.algorithms.modelgen (module), 409

nipype.caching.memory (module), 1552

nipype.conftest (module), 1554

nipype.interfaces.base.specs (module), 613

nipype.interfaces.base.support (module), 613

nipype.interfaces.base.traits_extension (module), 613

nipype.interfaces.brainsuite.brainsuite (module), 643

nipype.interfaces.cmtk.cmtk (module), 712

nipype.interfaces.cmtk.nbs (module), 716

nipype.interfaces.cmtk.nx (module), 718

nipype.interfaces.cmtk.parcellation (module), 720

nipype.interfaces.dcmstack (module), 1514

nipype.interfaces.dipy.preprocess (module), 733

nipype.interfaces.freesurfer.utils (module), 902

- nipy.interfaces.fsl.model (module), 984
- nipy.interfaces.io (module), 1535
- nipy.interfaces.matlab (module), 1554
- nipy.interfaces.minc.base (module), 1049
- nipy.interfaces.minc.testdata (module), 1095
- nipy.interfaces.mrtrix.convert (module), 1127
- nipy.interfaces.mrtrix.tensors (module), 1146
- nipy.interfaces.niftyreg.base (module), 1201
- nipy.interfaces.vtkbase (module), 1549
- nipy.pipeline.engine.base (module), 1565
- nipy.pipeline.engine.nodes (module), 1566
- nipy.pipeline.engine.utils (module), 1573
- nipy.pipeline.engine.workflows (module), 1576
- nipy.sphinxext.plot_workflow (module), 1579
- nipy.workflows.data (module), 3
- nipy.workflows.dmri.camino.connectivity_mapping (module), 5
- nipy.workflows.dmri.camino.diffusion (module), 6
- nipy.workflows.dmri.camino.group_connectivity (module), 7
- nipy.workflows.dmri.connectivity.group_connectivity (module), 8
- nipy.workflows.dmri.connectivity.nx (module), 11
- nipy.workflows.dmri.dipy.denoise (module), 13
- nipy.workflows.dmri.dti.tensor_registration (module), 14
- nipy.workflows.dmri.fsl.artifacts (module), 16
- nipy.workflows.dmri.fsl.dti (module), 38
- nipy.workflows.dmri.fsl.epi (module), 42
- nipy.workflows.dmri.fsl.tbss (module), 55
- nipy.workflows.dmri.fsl.utils (module), 69
- nipy.workflows.dmri.mrtrix.connectivity_mapping (module), 76
- nipy.workflows.dmri.mrtrix.diffusion (module), 77
- nipy.workflows.dmri.mrtrix.group_connectivity (module), 79
- nipy.workflows.fmri.fsl.estimate (module), 81
- nipy.workflows.fmri.fsl.preprocess (module), 86
- nipy.workflows.fmri.spm.preprocess (module), 98
- nipy.workflows.misc.utils (module), 105
- nipy.workflows.rsfmri.fsl.resting (module), 107
- nipy.workflows.smri.ants.ANTSBuildTemplate (module), 113
- nipy.workflows.smri.ants.antsRegistrationBuildTemplate (module), 115
- nipy.workflows.smri.freesurfer.autorecon1 (module), 116
- nipy.workflows.smri.freesurfer.autorecon2 (module), 116
- nipy.workflows.smri.freesurfer.bem (module), 116
- nipy.workflows.smri.freesurfer.recon (module), 118
- nipy.workflows.smri.freesurfer.utils (module), 120
- nipy.workflows.smri.niftyreg.groupwise (module), 126
- NitimeBaseInterface, 1246
- NlpFit, 1074
- Node (class in nipy.pipeline.engine.nodes), 1570
- nodelist_runner() (in module nipy.pipeline.engine.utils), 1574
- NonSteadyStateDetector, 373
- Norm, 1076
- Normalize, 829, 1468
- Normalize12, 1470
- NormalizeProbabilityMapSet, 398
- Notes, 531
- num_subnodes() (nipy.pipeline.engine.nodes.MapNode method), 1570
- NwarpAdjust, 532
- NwarpApply, 533
- NwarpCat, 535
- O**
- ODFRecon, 725
- ODFTracker, 727
- on_trait_change() (nipy.interfaces.matlab.MatlabInputSpec method), 1561
- on_trait_event() (nipy.interfaces.matlab.MatlabInputSpec method), 1562
- OneDToolPy, 536
- OneSampleTTest, 796
- OneSampleTTestDesign, 1453
- OrientScalarVolume, 1368
- OtsuThresholdImageFilter, 1400
- OtsuThresholdSegmentation, 1413
- out_of_date() (in module nipy.sphinxext.plot_workflow), 1581
- OutlierCount, 457
- output_dir() (nipy.pipeline.engine.nodes.JoinNode method), 1568
- output_dir() (nipy.pipeline.engine.nodes.MapNode method), 1570
- output_dir() (nipy.pipeline.engine.nodes.Node method), 1572
- output_spec (nipy.interfaces.matlab.MatlabCommand attribute), 1555
- outputs (nipy.pipeline.engine.base.EngineBase attribute), 1565
- outputs (nipy.pipeline.engine.nodes.JoinNode attribute), 1568
- outputs (nipy.pipeline.engine.nodes.MapNode attribute), 1570
- outputs (nipy.pipeline.engine.nodes.Node attribute), 1573
- outputs (nipy.pipeline.engine.workflows.Workflow attribute), 1578
- Overlap, 387, 399
- Overlay, 1030

P

[P2PDistance](#), 382
[package_version](#) (nipy.interfaces.matlab.MatlabInputSpec attribute), 1562
[Paint](#), 851
[PairedTTestDesign](#), 1455
[Parcellate](#), 719
[ParcellationStats](#), 881
[ParseDICOMDir](#), 830
[PatchMatch](#), 1232
[PercentileImage](#), 955
[PETPVC](#), 1543
[PETStandardUptakeValueComputation](#), 1415
[Pialmesh](#), 634
[PickAtlas](#), 400
[PicoPDFs](#), 673
[Pik](#), 1077
[PipeFunc](#) (class in nipy.caching.memory), 1553
[PlotMotionParams](#), 1032
[PlotTimeSeries](#), 1033
[PointsWarp](#), 771
[PowerSpectrum](#), 1034
[PRELUDE](#), 1010
[PrepareFieldmap](#), 934
[print_traits\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1562
[ProbabilisticSphericallyDeconvolutedStreamlineTrack](#), 1150
[ProbeVolumeWithModel](#), 1444
[ProbTrackX](#), 912
[ProbTrackX2](#), 915
[ProcStreamlines](#), 656
[ProjThresh](#), 919
[Pvc](#), 635
[pytest_unconfigure\(\)](#) (in module nipy.confest), 1554

Q

[QBallMX](#), 699
[QualityIndex](#), 458
[Quickshear](#), 1547
[Qwarp](#), 460
[QwarpPlusMinus](#), 469

R

[raise_exception\(\)](#) (nipy.interfaces.matlab.MatlabCommand method), 1555
[Randomise](#), 980
[RandomVol](#), 1121
[read_log\(\)](#) (in module nipy.caching.memory), 1554
[Realign](#), 1471
[RealignUnwarp](#), 1473
[ReconAll](#), 831
[references_](#) (nipy.interfaces.matlab.MatlabCommand attribute), 1555
[Refit](#), 540
[RegAladin](#), 1201
[RegAverage](#), 1207
[RegF3D](#), 1204
[Register](#), 852
[RegisterAVItoTairach](#), 853
[Registration](#), 563, 772
[RegistrationSynQuick](#), 573
[RegJacobian](#), 1209
[RegMeasure](#), 1210
[RegResample](#), 1211
[RegTools](#), 1212
[RegTransform](#), 1214
[ReHo](#), 538
[RelabelHypointensities](#), 883
[Remlfit](#), 423
[remove_coding\(\)](#) (in module nipy.sphinxext.plot_workflow), 1581
[remove_nodes\(\)](#) (nipy.pipeline.engine.workflows.Workflow method), 1578
[remove_trait\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1562
[remove_trait_listener\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1562
[RemoveIntersection](#), 884
[RemoveNeck](#), 884
[Rename](#), 1491
[render_figures\(\)](#) (in module nipy.sphinxext.plot_workflow), 1581
[Reorient](#), 1517
[Reorient2Std](#), 1035
[ReportCapableInterface](#), 1123
[Resample](#), 542, 733, 836, 1079
[ResampleDTIVolume](#), 1376
[ResampleScalarVectorDWIVolume](#), 1394
[ResampleScalarVolume](#), 1401
[Rescale](#), 1518
[reset_traits\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1562
[Reshape](#), 1084
[Reslice](#), 1486
[ResliceToReference](#), 1487
[resource_monitor](#) (nipy.interfaces.matlab.MatlabCommand attribute), 1555
[ResponseSD](#), 1165
[RESTORE](#), 736
[result](#) (nipy.pipeline.engine.nodes.JoinNode attribute), 1568
[result](#) (nipy.pipeline.engine.nodes.MapNode attribute), 1570
[result](#) (nipy.pipeline.engine.nodes.Node attribute), 1573
[Retroicor](#), 479
[Rigid](#), 753

- RigidRegistration, 1411
- RigidTask, 754
- rm_all_but() (in module nipy.caching.memory), 1554
- RobustFOV, 1036
- RobustRegister, 837
- RobustStatisticsSegmenter, 1437
- RobustTemplate, 776
- ROIGen, 711
- ROIStats, 477
- run() (in module nipy.sphinxext.plot_workflow), 1581
- run() (nipy.interfaces.matlab.MatlabCommand class method), 1555
- run() (nipy.pipeline.engine.nodes.JoinNode method), 1568
- run() (nipy.pipeline.engine.nodes.MapNode method), 1570
- run() (nipy.pipeline.engine.nodes.Node method), 1573
- run() (nipy.pipeline.engine.workflows.Workflow method), 1578
- run_code() (in module nipy.sphinxext.plot_workflow), 1581
- S**
- S3DataGrabber, 1530
- SampleToSurface, 885
- save() (nipy.pipeline.engine.base.EngineBase method), 1565
- save() (nipy.pipeline.engine.nodes.JoinNode method), 1568
- save() (nipy.pipeline.engine.nodes.MapNode method), 1570
- save() (nipy.pipeline.engine.nodes.Node method), 1573
- save() (nipy.pipeline.engine.workflows.Workflow method), 1578
- save_hashfile() (in module nipy.pipeline.engine.utils), 1574
- save_inputs_to_json() (nipy.interfaces.matlab.MatlabCommand class method), 1556
- save_resultfile() (in module nipy.pipeline.engine.utils), 1574
- scalartransform, 1314
- Scrubmask, 639
- Seg, 481
- Segment, 1475
- SegmentCC, 839
- SegmentWM, 840
- SegStats, 800
- SegStatsReconAll, 803
- Select, 1492
- SelectFiles, 1532
- SEMLikeCommandLine, 611
- set() (nipy.interfaces.matlab.MatlabInputSpec method), 1562
- set_default_matlab_cmd() (nipy.interfaces.matlab.MatlabCommand class method), 1556
- set_default_mfile() (nipy.interfaces.matlab.MatlabCommand class method), 1556
- set_default_paths() (nipy.interfaces.matlab.MatlabCommand class method), 1556
- set_default_terminal_output() (nipy.interfaces.matlab.MatlabCommand class method), 1556
- set_input() (nipy.pipeline.engine.nodes.JoinNode method), 1568
- set_input() (nipy.pipeline.engine.nodes.MapNode method), 1570
- set_input() (nipy.pipeline.engine.nodes.Node method), 1573
- set_trait_dispatch_handler() (nipy.interfaces.matlab.MatlabInputSpec class method), 1563
- setup() (in module nipy.sphinxext.plot_workflow), 1582
- SFLUTGen, 645
- SFPeaks, 701
- SFPICOCalibData, 647
- Shredder, 659
- ShuffleVectorsModule, 1363
- SigLoss, 935, 1037
- SignalExtraction, 1541
- Similarity, 388, 1243
- SimilarityIndex, 1250
- SimpleInterface, 611
- SimpleRegionGrowingSegmentation, 1433
- SimpleThreshold, 400
- SimulateMultiTensor, 737
- Skullfinder, 640
- SkullStrip, 482
- Slice, 1038
- Slicer, 1038
- SlicerCommandLine, 1367, 1515
- SliceTimer, 1013
- SliceTiming, 1477
- SMM, 983
- Smooth, 841, 1040, 1478
- SmoothEstimate, 983
- SmoothTessellation, 888
- SpaceTimeRealigner, 1241
- SpatialFilter, 956
- SpecifyModel, 405
- SpecifySparseModel, 408
- SpecifySPMModel, 407
- Sphere, 890
- SphericalAverage, 807
- SphericallyDeconvolutedStreamlineTrack, 1153
- SplineFilter, 729

Split, 1041, 1492
 SplitNifti, 1514
 SplitROIs, 400
 SQLiteSink, 1530
 SSHDataGrabber, 1531
 STAPLEAnalysis, 1312
 StatsCommand, 1235
 StdImage, 957
 StdOutCommandLine, 612
 StimulusCorrelation, 415
 StreamlineTrack, 1155
 StreamlineTractography, 739
 strip_temp() (in module nipype.pipeline.engine.utils), 1575
 SubtractScalarVolumes, 1382
 Surface2VolTransform, 891
 SurfaceSmooth, 892
 SurfaceSnapshots, 893
 SurfaceTransform, 896
 SUSAN, 1012
 SVAdjustVoxSp, 760
 SVAdjustVoxSpTask, 761
 SVMTest, 498
 SVMTrain, 499
 SVReg, 636
 SVResample, 761
 SVResampleTask, 762
 SwapDimensions, 1042
 sync_trait() (nipype.interfaces.matlab.MatlabInputSpec method), 1563
 synchronize_iterables() (in module nipype.pipeline.engine.utils), 1575
 Synthesize, 428
 SynthesizeFLASH, 842

T

TalairachAVI, 897
 TalairachQC, 898
 Tca, 641
 TCat, 543
 TCatSubBrick, 544
 TCK2VTK, 1183
 TCompCor, 374
 TCorr1D, 483
 TCorrelate, 486
 TCorrMap, 484
 TemporalFilter, 958
 Tensor2ApparentDiffusion, 1135
 Tensor2FractionalAnisotropy, 1136
 Tensor2Vector, 1137
 TensorMetrics, 1184
 TensorMode, 739
 terminal_output (nipype.interfaces.matlab.MatlabCommand attribute), 1556
 TextureFromNoiseImageFilter, 1313
 TextureMeasureFilter, 1314
 ThicknessPVC, 642
 Threshold, 959, 1138, 1456
 ThresholdScalarVolume, 1396
 ThresholdStatistics, 1457
 Tkregister2, 899
 TNorm, 487
 To3D, 546
 ToEcat, 1085
 topological_sort() (in module nipype.pipeline.engine.utils), 1575
 TOPUP, 936
 ToRaw, 1086
 TProject, 489
 Track, 674
 TrackBallStick, 677
 TrackBayesDirac, 679
 TrackBedpostxDeter, 682
 TrackBedpostxProba, 685
 TrackBootstrap, 688
 TrackDensityMap, 740
 TrackDT, 691
 TrackMerge, 730
 TrackPICO, 693
 Tracks2Prob, 1157
 Trackvis2Camino, 706
 Tractography, 1170
 TractographyLabelMapSeeding, 1378
 TractShredder, 660
 TractSkeleton, 920
 Training, 941
 TrainingSetCreator, 941
 trait() (nipype.interfaces.matlab.MatlabInputSpec method), 1563
 trait_context() (nipype.interfaces.matlab.MatlabInputSpec method), 1563
 trait_get() (nipype.interfaces.matlab.MatlabInputSpec method), 1563
 trait_items_event() (nipype.interfaces.matlab.MatlabInputSpec method), 1563
 trait_monitor() (nipype.interfaces.matlab.MatlabInputSpec class method), 1563
 trait_names() (nipype.interfaces.matlab.MatlabInputSpec method), 1563
 trait_property_changed() (nipype.interfaces.matlab.MatlabInputSpec method), 1563
 trait_set() (nipype.interfaces.matlab.MatlabInputSpec method), 1563
 trait_setq() (nipype.interfaces.matlab.MatlabInputSpec method), 1563
 trait_subclasses() (nipype.interfaces.matlab.MatlabInputSpec class method), 1564

[trait_view\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1564
[trait_view_elements\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1564
[trait_views\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1564
[traits\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1564
[traits_init\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1564
[traits_inited\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1564
[Trim](#), 1242
[TShift](#), 492
[TSNR](#), 375, 401
[TStat](#), 545
[TupleMaths](#), 1229
[TVAdjustOriginTask](#), 763
[TVAdjustVoxSp](#), 764
[TVAdjustVoxSpTask](#), 765
[TVResample](#), 765
[TVResampleTask](#), 766
[TVTKBaseInterface](#), 383
[TVtool](#), 767
[TVtoolTask](#), 768
[TwoSampleTTestDesign](#), 1458

U

[UKFTractography](#), 1297
[UnaryMaths](#), 960, 1230
[UnaryStats](#), 1236
[UnbiasedNonLocalMeans](#), 1300
[Undump](#), 547
[unescape_doctest\(\)](#) (in module nipy.sphinxext.plot_workflow), 1582
[Unifize](#), 549
[UnpackSDICOMDir](#), 843
[update\(\)](#) (nipy.pipeline.engine.nodes.JoinNode method), 1568
[update\(\)](#) (nipy.pipeline.engine.nodes.MapNode method), 1570
[update\(\)](#) (nipy.pipeline.engine.nodes.Node method), 1573

V

[validate_trait\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1564
[VBMSegment](#), 1479
[VBRAINSDemonWarp](#), 1330, 1429
[VecReg](#), 921
[version](#) (nipy.interfaces.matlab.MatlabCommand attribute), 1556
[version_from_command\(\)](#) (nipy.interfaces.matlab.MatlabCommand

method), 1556
[visible_traits\(\)](#) (nipy.interfaces.matlab.MatlabInputSpec method), 1564
[Vnifti2Image](#), 1495
[Volcentre](#), 1089
[Voliso](#), 1090
[Volpad](#), 1091
[Volreg](#), 494
[VolSymm](#), 1088
[VolumeMask](#), 901
[VotingBinaryHoleFillingImageFilter](#), 1397
[VtkStreamlines](#), 661
[VtoMat](#), 1496

W

[walk\(\)](#) (in module nipy.pipeline.engine.utils), 1575
[walk_files\(\)](#) (in module nipy.pipeline.engine.utils), 1575
[walk_outputs\(\)](#) (in module nipy.pipeline.engine.utils), 1575
[Warp](#), 496
[WarpImageMultiTransform](#), 578
[WarpPoints](#), 383, 1043
[WarpPointsFromStd](#), 1044
[WarpPointsToStd](#), 1045
[WarpTimeSeriesImageMultiTransform](#), 580
[WarpUtils](#), 1046
[WatershedBEM](#), 1125
[WatershedSkullStrip](#), 845
[WBCommand](#), 1497
[wf_directive\(\)](#) (in module nipy.sphinxext.plot_workflow), 1582
[with_traceback\(\)](#) (nipy.sphinxext.plot_workflow.GraphError method), 1581
[Workflow](#) (class in nipy.pipeline.engine.workflows), 1576
[wrappers](#) (nipy.interfaces.matlab.MatlabInputSpec attribute), 1564
[write_graph\(\)](#) (nipy.pipeline.engine.workflows.Workflow method), 1578
[write_hierarchical_dotfile\(\)](#) (nipy.pipeline.engine.workflows.Workflow method), 1578
[write_report\(\)](#) (in module nipy.pipeline.engine.utils), 1575
[write_workflow_prov\(\)](#) (in module nipy.pipeline.engine.utils), 1575
[write_workflow_resources\(\)](#) (in module nipy.pipeline.engine.utils), 1575

X

[XFibres5](#), 923
[XfmAvg](#), 1092
[XfmConcat](#), 1093

XfmInvert, [1094](#)
XNATSink, [1533](#)
XNATSource, [1534](#)

Z

Zcat, [552](#)
ZCutUp, [551](#)
Zeropad, [553](#)