# CONTENTS

## I  User Guide

### 1 Introduction
1.1 What is NIPY for? .............................................. 3
1.2 A history of NIPY .............................................. 3

### 2 Download and Install
2.1 Dependencies for install ........................................ 5
2.2 Building from source code ...................................... 7

### 3 Geography of the Scipy world
3.1 Numpy ......................................................... 9
3.2 SciPy ......................................................... 9
3.3 Matplotlib ..................................................... 9
3.4 Ipython ....................................................... 10
3.5 Cython ....................................................... 10
3.6 Mayavi ....................................................... 10

### 4 Tutorials
4.1 Basic Data IO .................................................. 11
4.2 Basics of the Coordinate Map .................................. 12
4.3 Specifying a GLM in NiPy ...................................... 15
4.4 Neuronal model ................................................ 22
4.5 Events with amplitudes ........................................ 26
4.6 Design matrix ................................................ 29
4.7 Formula objects .............................................. 30

### 5 Glossary ....................................................... 31

## II  NeuroSpin tools

### 6 Mask-extraction utilities
6.1 nipy.labs.utils.mask.compute_mask .......................... 37
6.2 nipy.labs.utils.mask.compute_mask_files .................. 38
6.3 nipy.labs.utils.mask.compute_mask_sessions ............... 39

### 7 Empirical null
7.1 Example ..................................................... 41
7.2 Class documentation ......................................... 42
<table>
<thead>
<tr>
<th>Module</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>algorithms.clustering.utils</code></td>
<td>213</td>
</tr>
<tr>
<td>25.1 Module: <code>algorithms.clustering.utils</code></td>
<td>213</td>
</tr>
<tr>
<td>25.2 Functions</td>
<td>213</td>
</tr>
<tr>
<td><code>algorithms.clustering.von_mises_fisher_mixture</code></td>
<td>215</td>
</tr>
<tr>
<td>26.1 Module: <code>algorithms.clustering.von_mises_fisher_mixture</code></td>
<td>215</td>
</tr>
<tr>
<td>26.2 Class</td>
<td>215</td>
</tr>
<tr>
<td>26.3 VonMisesMixture</td>
<td>215</td>
</tr>
<tr>
<td>26.4 Functions</td>
<td>218</td>
</tr>
<tr>
<td><code>algorithms.diagnostics.screens</code></td>
<td>221</td>
</tr>
<tr>
<td>27.1 Module: <code>algorithms.diagnostics.screens</code></td>
<td>221</td>
</tr>
<tr>
<td>27.2 Functions</td>
<td>221</td>
</tr>
<tr>
<td><code>algorithms.diagnostics.timediff</code></td>
<td>223</td>
</tr>
<tr>
<td>28.1 Module: <code>algorithms.diagnostics.timediff</code></td>
<td>223</td>
</tr>
<tr>
<td><code>algorithms.diagnostics.tsdiffplot</code></td>
<td>225</td>
</tr>
<tr>
<td>29.1 Module: <code>algorithms.diagnostics.tsdiffplot</code></td>
<td>225</td>
</tr>
<tr>
<td>29.2 Functions</td>
<td>225</td>
</tr>
<tr>
<td><code>algorithms.fwhm</code></td>
<td>227</td>
</tr>
<tr>
<td>30.1 Module: <code>algorithms.fwhm</code></td>
<td>227</td>
</tr>
<tr>
<td>30.2 Classes</td>
<td>227</td>
</tr>
<tr>
<td><code>algorithms.graph.bipartite_graph</code></td>
<td>231</td>
</tr>
<tr>
<td>31.1 Module: <code>algorithms.graph.bipartite_graph</code></td>
<td>231</td>
</tr>
<tr>
<td>31.2 Class</td>
<td>231</td>
</tr>
<tr>
<td>31.3 BipartiteGraph</td>
<td>231</td>
</tr>
<tr>
<td>31.4 Functions</td>
<td>233</td>
</tr>
<tr>
<td><code>algorithms.graph.field</code></td>
<td>235</td>
</tr>
<tr>
<td>32.1 Module: <code>algorithms.graph.field</code></td>
<td>235</td>
</tr>
<tr>
<td>32.2 Class</td>
<td>235</td>
</tr>
<tr>
<td>32.3 Field</td>
<td>235</td>
</tr>
<tr>
<td>32.4 Functions</td>
<td>244</td>
</tr>
<tr>
<td><code>algorithms.graph.forest</code></td>
<td>245</td>
</tr>
<tr>
<td>33.1 Module: <code>algorithms.graph.forest</code></td>
<td>245</td>
</tr>
<tr>
<td>33.2 Forest</td>
<td>245</td>
</tr>
<tr>
<td><code>algorithms.graph.graph</code></td>
<td>255</td>
</tr>
<tr>
<td>34.1 Module: <code>algorithms.graph.graph</code></td>
<td>255</td>
</tr>
<tr>
<td>34.2 Classes</td>
<td>255</td>
</tr>
<tr>
<td>34.3 Functions</td>
<td>263</td>
</tr>
<tr>
<td><code>algorithms.interpolation</code></td>
<td>265</td>
</tr>
<tr>
<td>35.1 Module: <code>algorithms.interpolation</code></td>
<td>265</td>
</tr>
<tr>
<td>35.2 ImageInterpolator</td>
<td>265</td>
</tr>
<tr>
<td><code>algorithms.kernel_smooth</code></td>
<td>267</td>
</tr>
<tr>
<td>36.1 Module: <code>algorithms.kernel_smooth</code></td>
<td>267</td>
</tr>
<tr>
<td>36.2 Class</td>
<td>267</td>
</tr>
<tr>
<td>36.3 LinearFilter</td>
<td>267</td>
</tr>
<tr>
<td>36.4 Functions</td>
<td>268</td>
</tr>
<tr>
<td>Module</td>
<td>Page</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>algorithms.optimize</td>
<td>271</td>
</tr>
<tr>
<td>37.1 Module: algorithms.optimize</td>
<td></td>
</tr>
<tr>
<td>37.1.1 ChainTransform</td>
<td>279</td>
</tr>
<tr>
<td>algorithms.registration.affine</td>
<td>273</td>
</tr>
<tr>
<td>38.1 Module: algorithms.registration.affine</td>
<td></td>
</tr>
<tr>
<td>38.1.1 ChainTransform</td>
<td>279</td>
</tr>
<tr>
<td>algorithms.registration.chain_transform</td>
<td>281</td>
</tr>
<tr>
<td>39.1 Module: algorithms.registration.chain_transform</td>
<td></td>
</tr>
<tr>
<td>39.1.1 ChainTransform</td>
<td>281</td>
</tr>
<tr>
<td>39.2 Function</td>
<td>281</td>
</tr>
<tr>
<td>algorithms.registration.groupwise_registration</td>
<td>283</td>
</tr>
<tr>
<td>40.1 Module: algorithms.registration.groupwise_registration</td>
<td></td>
</tr>
<tr>
<td>40.1.1 ChainTransform</td>
<td>281</td>
</tr>
<tr>
<td>40.2 Function</td>
<td>283</td>
</tr>
<tr>
<td>algorithms.registration.histogram_registration</td>
<td>291</td>
</tr>
<tr>
<td>41.1 Module: algorithms.registration.histogram_registration</td>
<td></td>
</tr>
<tr>
<td>41.1.1 ChainTransform</td>
<td>291</td>
</tr>
<tr>
<td>41.2 Class</td>
<td>291</td>
</tr>
<tr>
<td>41.3 Function</td>
<td>293</td>
</tr>
<tr>
<td>algorithms.registration.optimizer</td>
<td>295</td>
</tr>
<tr>
<td>42.1 Module: algorithms.registration.optimizer</td>
<td></td>
</tr>
<tr>
<td>42.1.1 ChainTransform</td>
<td>291</td>
</tr>
<tr>
<td>42.1.2 Function</td>
<td>293</td>
</tr>
<tr>
<td>algorithms.registration.polyaffine</td>
<td>297</td>
</tr>
<tr>
<td>43.1 Module: algorithms.registration.polyaffine</td>
<td></td>
</tr>
<tr>
<td>43.1.1 ChainTransform</td>
<td>297</td>
</tr>
<tr>
<td>43.2 Function</td>
<td>297</td>
</tr>
<tr>
<td>algorithms.registration.resample</td>
<td>299</td>
</tr>
<tr>
<td>44.1 Module: algorithms.registration.resample</td>
<td></td>
</tr>
<tr>
<td>44.1.1 ChainTransform</td>
<td>299</td>
</tr>
<tr>
<td>45.1 Module: algorithms.registration.similarity_measures</td>
<td>301</td>
</tr>
<tr>
<td>45.1.1 ChainTransform</td>
<td>301</td>
</tr>
<tr>
<td>45.1.1 Class</td>
<td>306</td>
</tr>
<tr>
<td>46.1 Module: algorithms.registration.transform</td>
<td>307</td>
</tr>
<tr>
<td>46.1.1 ChainTransform</td>
<td>307</td>
</tr>
<tr>
<td>46.2 Function</td>
<td>307</td>
</tr>
<tr>
<td>algorithms.resample</td>
<td>309</td>
</tr>
<tr>
<td>47.1 Module: algorithms.resample</td>
<td></td>
</tr>
<tr>
<td>47.1.1 ChainTransform</td>
<td>309</td>
</tr>
<tr>
<td>47.2 Function</td>
<td>309</td>
</tr>
<tr>
<td>algorithms.segmentation.brain_segmentation</td>
<td>311</td>
</tr>
<tr>
<td>48.1 Module: algorithms.segmentation.brain_segmentation</td>
<td></td>
</tr>
<tr>
<td>48.1.1 ChainTransform</td>
<td>311</td>
</tr>
<tr>
<td>48.2 Function</td>
<td>311</td>
</tr>
<tr>
<td>algorithms.segmentation.segmentation</td>
<td>313</td>
</tr>
<tr>
<td>49.1 Module: algorithms.segmentation.segmentation</td>
<td></td>
</tr>
<tr>
<td>49.1.1 ChainTransform</td>
<td>313</td>
</tr>
<tr>
<td>49.2 Class</td>
<td>313</td>
</tr>
<tr>
<td>49.3 Function</td>
<td>313</td>
</tr>
<tr>
<td>49.4 Function</td>
<td>314</td>
</tr>
</tbody>
</table>
50 algorithms.statistics.bench.bench_intvol
   50.1 Module: algorithms.statistics.bench.bench_intvol .................. 317
   50.2 Functions ......................................................... 317

51 algorithms.statistics.empirical_pvalue
   51.1 Module: algorithms.statistics.empirical_pvalue ..................... 319
   51.2 Class .............................................................. 319
   51.3 NormalEmpiricalNull ............................................. 319
   51.4 Functions .......................................................... 321

52 algorithms.statistics.formula.formulae
   52.1 Module: algorithms.statistics.formula.formulae ..................... 325
   52.2 Classes .............................................................. 327
   52.3 Functions ............................................................ 417

53 algorithms.statistics.mixed_effects_stat
   53.1 Module: algorithms.statistics.mixed_effects_stat ................. 423
   53.2 Class .............................................................. 423
   53.3 MixedEffectsModel ................................................. 423
   53.4 Functions ............................................................ 424

54 algorithms.statistics.models.family.family
   54.1 Module: algorithms.statistics.models.family.family ............... 429
   54.2 Classes .............................................................. 429

55 algorithms.statistics.models.family.links
   55.1 Module: algorithms.statistics.models.family.links ............... 439
   55.2 Classes .............................................................. 439

56 algorithms.statistics.models.family.varfuncs
   56.1 Module: algorithms.statistics.models.family.varfuncs ............ 445
   56.2 Classes .............................................................. 445

57 algorithms.statistics.models(glm
   57.1 Module: algorithms.statistics.models glm ........................... 447
   57.2 Model ............................................................... 447

58 algorithms.statistics.models.model
   58.1 Module: algorithms.statistics.models.model ....................... 451
   58.2 Classes .............................................................. 451

59 algorithms.statistics.models.nlsmodel
   59.1 Module: algorithms.statistics.models.nlsmodel ................... 457
   59.2 NLSModel ............................................................ 457

60 algorithms.statistics.models.regression
   60.1 Module: algorithms.statistics.models.regression .................. 461
   60.2 Classes .............................................................. 461
   60.3 Functions ............................................................ 478

61 algorithms.statistics.models.setupscons
   61.1 Module: algorithms.statistics.models.setupscons .................. 481

62 algorithms.statistics.models.utils
   62.1 Module: algorithms.statistics.models.utils ........................ 483
   62.2 Class ............................................................... 483
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>75.2</td>
<td>Functions</td>
</tr>
<tr>
<td>76</td>
<td>core.reference.spaces</td>
</tr>
<tr>
<td>76.1</td>
<td>Module: core.reference.spaces</td>
</tr>
<tr>
<td>76.2</td>
<td>Classes</td>
</tr>
<tr>
<td>76.3</td>
<td>Functions</td>
</tr>
<tr>
<td>77</td>
<td>core.utils.generators</td>
</tr>
<tr>
<td>77.1</td>
<td>Module: core.utils.generators</td>
</tr>
<tr>
<td>77.2</td>
<td>Functions</td>
</tr>
<tr>
<td>78</td>
<td>interfaces.matlab</td>
</tr>
<tr>
<td>78.1</td>
<td>Module: interfaces.matlab</td>
</tr>
<tr>
<td>78.2</td>
<td>Functions</td>
</tr>
<tr>
<td>79</td>
<td>interfaces.spm</td>
</tr>
<tr>
<td>79.1</td>
<td>Module: interfaces.spm</td>
</tr>
<tr>
<td>79.2</td>
<td>Class</td>
</tr>
<tr>
<td>79.3</td>
<td>SpmInfo</td>
</tr>
<tr>
<td>79.4</td>
<td>Functions</td>
</tr>
<tr>
<td>80</td>
<td>io.files</td>
</tr>
<tr>
<td>80.1</td>
<td>Module: io.files</td>
</tr>
<tr>
<td>80.2</td>
<td>Functions</td>
</tr>
<tr>
<td>81</td>
<td>io.nifti_ref</td>
</tr>
<tr>
<td>81.1</td>
<td>Module: io.nifti_ref</td>
</tr>
<tr>
<td>81.2</td>
<td>Class</td>
</tr>
<tr>
<td>81.3</td>
<td>NiftiError</td>
</tr>
<tr>
<td>81.4</td>
<td>Functions</td>
</tr>
<tr>
<td>82</td>
<td>labs.bindings.benchmarks.bench_numpy</td>
</tr>
<tr>
<td>82.1</td>
<td>Module: labs.bindings.benchmarks.bench_numpy</td>
</tr>
<tr>
<td>82.2</td>
<td>Functions</td>
</tr>
<tr>
<td>83</td>
<td>labs.datasets.converters</td>
</tr>
<tr>
<td>83.1</td>
<td>Module: labs.datasets.converters</td>
</tr>
<tr>
<td>83.2</td>
<td>Functions</td>
</tr>
<tr>
<td>84</td>
<td>labs.datasets.transforms.affine_transform</td>
</tr>
<tr>
<td>84.1</td>
<td>Module: labs.datasets.transforms.affine_transform</td>
</tr>
<tr>
<td>84.2</td>
<td>AffineTransform</td>
</tr>
<tr>
<td>85</td>
<td>labs.datasets.transforms.affine_utils</td>
</tr>
<tr>
<td>85.1</td>
<td>Module: labs.datasets.transforms.affine_utils</td>
</tr>
<tr>
<td>85.2</td>
<td>Functions</td>
</tr>
<tr>
<td>86</td>
<td>labs.datasets.transforms.transform</td>
</tr>
<tr>
<td>86.1</td>
<td>Module: labs.datasets.transforms.transform</td>
</tr>
<tr>
<td>86.2</td>
<td>Classes</td>
</tr>
<tr>
<td>87</td>
<td>labs.datasets.volumes.volume_data</td>
</tr>
<tr>
<td>87.1</td>
<td>Module: labs.datasets.volumes.volume_data</td>
</tr>
<tr>
<td>87.2</td>
<td>VolumeData</td>
</tr>
<tr>
<td>Module Name</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>labs.viz_tools.cm</td>
<td>735</td>
</tr>
<tr>
<td>114.1 Module: labs.viz_tools.cm</td>
<td>735</td>
</tr>
<tr>
<td>114.2 Functions</td>
<td>735</td>
</tr>
<tr>
<td>labs.viz_tools.coord_tools</td>
<td>737</td>
</tr>
<tr>
<td>115.1 Module: labs.viz_tools.coord_tools</td>
<td>737</td>
</tr>
<tr>
<td>115.2 Functions</td>
<td>737</td>
</tr>
<tr>
<td>labs.viz_tools.maps_3d</td>
<td>739</td>
</tr>
<tr>
<td>116.1 Module: labs.viz_tools.maps_3d</td>
<td>739</td>
</tr>
<tr>
<td>116.2 Functions</td>
<td>739</td>
</tr>
<tr>
<td>labs.viz_tools.slicers</td>
<td>743</td>
</tr>
<tr>
<td>117.1 Module: labs.viz_tools.slicers</td>
<td>743</td>
</tr>
<tr>
<td>117.2 Classes</td>
<td>743</td>
</tr>
<tr>
<td>117.3 Function</td>
<td>759</td>
</tr>
<tr>
<td>labs.viz_tools.test.test_activation_maps</td>
<td>761</td>
</tr>
<tr>
<td>118.1 Module: labs.viz_tools.test.test_activation_maps</td>
<td>761</td>
</tr>
<tr>
<td>118.2 Functions</td>
<td>761</td>
</tr>
<tr>
<td>labs.viz_tools.test.test_cm</td>
<td>763</td>
</tr>
<tr>
<td>119.1 Module: labs.viz_tools.test.test_cm</td>
<td>763</td>
</tr>
<tr>
<td>119.2 Functions</td>
<td>763</td>
</tr>
<tr>
<td>labs.viz_tools.test.test_coord_tools</td>
<td>765</td>
</tr>
<tr>
<td>120.1 Module: labs.viz_tools.test.test_coord_tools</td>
<td>765</td>
</tr>
<tr>
<td>120.2 Functions</td>
<td>765</td>
</tr>
<tr>
<td>labs.viz_tools.test.test_edge_detect</td>
<td>767</td>
</tr>
<tr>
<td>121.1 Module: labs.viz_tools.test.test_edge_detect</td>
<td>767</td>
</tr>
<tr>
<td>121.2 Functions</td>
<td>767</td>
</tr>
<tr>
<td>labs.viz_tools.test.test_slices</td>
<td>769</td>
</tr>
<tr>
<td>122.1 Module: labs.viz_tools.test.test_slices</td>
<td>769</td>
</tr>
<tr>
<td>modalities.fmri.design</td>
<td>771</td>
</tr>
<tr>
<td>123.1 Module: modalities.fmri.design</td>
<td>771</td>
</tr>
<tr>
<td>123.2 Functions</td>
<td>771</td>
</tr>
<tr>
<td>modalities.fmri.design_matrix</td>
<td>775</td>
</tr>
<tr>
<td>124.1 Module: modalities.fmri.design_matrix</td>
<td>775</td>
</tr>
<tr>
<td>124.2 Class</td>
<td>775</td>
</tr>
<tr>
<td>124.3 DesignMatrix</td>
<td>775</td>
</tr>
<tr>
<td>124.4 Functions</td>
<td>776</td>
</tr>
<tr>
<td>modalities.fmri.experimental_paradigm</td>
<td>779</td>
</tr>
<tr>
<td>125.1 Module: modalities.fmri.experimental_paradigm</td>
<td>779</td>
</tr>
<tr>
<td>125.2 Classes</td>
<td>779</td>
</tr>
<tr>
<td>125.3 Function</td>
<td>781</td>
</tr>
<tr>
<td>modalities.fmri.fmri</td>
<td>783</td>
</tr>
<tr>
<td>126.1 Module: modalities.fmri.fmri</td>
<td>783</td>
</tr>
<tr>
<td>126.2 FmriImageList</td>
<td>783</td>
</tr>
</tbody>
</table>
Part I

User Guide
INTRODUCTION

As you can see, we do not yet have much of a user guide for NIPY. We are spending all our effort in developing the building blocks of the code, and we have not yet returned to a guide to how to use it.

We are starting to write general Tutorials, that include introductions to how to use NIPY code to run analyses.

1.1 What is NIPY for?

The purpose of NIPY is to make it easier to do better brain imaging research. We believe that neuroscience ideas and analysis ideas develop together. Good ideas come from understanding; understanding comes from clarity, and clarity must come from well-designed teaching materials and well-designed software. The software must be designed as a natural extension of the underlying ideas.

We aim to build software that is:

• clearly written
• clearly explained
• a good fit for the underlying ideas
• a natural home for collaboration

We hope that, if we fail to do this, you will let us know. We will try and make it better.

The NIPY team

1.2 A history of NIPY

Sometime around 2002, Jonathan Taylor started writing BrainSTAT, a Python version of Keith Worsley’s FmriSTAT package.

In 2004, Jarrod Millman and Matthew Brett decided that they wanted to write a grant to build a new neuroimaging analysis package in Python. Soon afterwards, they found that Jonathan had already started, and merged efforts. At first we called this project BrainPy. Later we changed the name to NIPY.

In 2005, Jarrod, Matthew and Jonathan, along with Mark D’Esposito, Fernando Perez, John Hunter, Jean-Baptiste Poline, and Tom Nichols, submitted the first NIPY grant to the NIH. It was not successful.

In 2006, Jarrod and Mark submitted a second grant, based on the first. The NIH gave us 3 years of funding for two programmers. We hired two programmers in 2007 - Christopher Burns and Tom Waite - and began work on refactoring the code.
Meanwhile, the team at Neurospin, Paris, started to refactor their FFF code to work better with python and NIPY. This work was by Alexis Roche, Bertrand Thirion, and Benjamin Thyreau, with some help and advice from Fernando Perez.

In 2008, Fernando Perez and Matthew Brett started work full-time at the UC Berkeley Brain Imaging Center. Matthew in particular came to work on NIPY.
DOWNLOAD AND INSTALL

This page covers the necessary steps to install and run NIPY. Below is a list of required dependencies, along with additional software recommendations.

2.1 Dependencies for install

2.1.1 Must Have

- **Python** 2.5 or later
- **NumPy** 1.2 or later: Numpy is an array library for Python
- **SciPy** 0.7 or later: Scipy contains scientific computing libraries based on numpy
- **Sympy** 0.6.6 or later: Sympy is a symbolic mathematics library for Python. We use it for statistical formulae.

2.1.2 Strong Recommendations

- **IPython**: Interactive Python environment.
- **Matplotlib**: python plotting library.

2.1.3 Installing from binary packages

For Debian or Ubuntu

Please use the NeuroDebian repository, and install with:

```
sudo apt-get install python-nipy
```

This will install the dependencies for you.

For Fedora, CentOS

```
sudo yum install numpy scipy sympy python-setuptools
sudo yum install python-devel gcc
sudo easy_install nibabel
sudo easy_install nipy
```
For OSX

Install Python, Numpy, and Scipy via their respective dmg installers.
Install via distribute / setuptools and easy_install. See the distribute page for how to install easy_install and related tools. Then (from the command prompt):

easy_install nipy

For Windows

Option 1  You can make your life much easier by using Python (X, Y). This will install Python, Numpy, Scipy, IPython, Matplotlib, Sympy and many other useful things.
Then go to nipy pypi and download the .exe installer for nipy. Double click to install.

Option 2  
- Download Python and install with the exe or msi installer
- Download and install the “Scipy stack” from Christophe Gohlke’s unofficial windows binaries.
- If the nipy version on the unofficial windows binaries page is current, use that, otherwise, go to nipy pypi, download and install the exe installer for nipy

Option 3  Consider one of the big Python bundles such as EPD free or Anaconda CE for the dependencies. Install nipy from the exe installer at nipy pypi.

Option 4  Do all the installs by hand:
- Download Python and install with the exe or msi installer. Make sure your python and the scripts directory (say, c:\Python27\Scripts) are on your windows path.
- Download Numpy and Scipy exe installers for your Python version from their respective Numpy and Scipy download sites.
- Install distribute to give you easy_install.
- Install pip using easy_install from a windows cmd shell:
  
easy_install pip

- Install sympy and nibabel using pip from a window cmd shell:
  
pip install sympy
  pip install nibabel

- On 32-bit Windows, install nipy using easy_install:
  
easy_install nipy

  This will pick up and use the exe installer. For 64-bits install use the installer at the unofficial windows binaries site.

Otherwise

I’m afraid you might need to build from source...
2.2 Building from source code

2.2.1 Dependencies for build

- A C compiler: NIPY does contain a few C extensions for optimized routines. Therefore, you must have a compiler to build from source. XCode (OSX) and MinGW (Windows) both include a C compiler. On Linux, try `sudo apt-get build-essential` on Debian / Ubuntu, `sudo yum install gcc` on Fedora and related distributions.

2.2.2 Recommended for build

- Cython 0.12.1 or later: Cython is a language that is a fusion of Python and C. It allows us to write fast code using Python and C syntax, so that it easier to read and maintain. You don’t need it to build a release, unless you modify the Cython *.pyx files in the nipy distribution.

2.2.3 Procedure

Developers should look through the development quickstart documentation. There you will find information on building NIPY, the required software packages and our developer guidelines.

If you are primarily interested in using NIPY, download the source tarball from nipy pypi and follow these instructions for building. The installation process is similar to other Python packages so it will be familiar if you have Python experience.

Unpack the source tarball and change into the source directory. Once in the source directory, you can build the neuroimaging package using:

```
python setup.py build
```

To install, simply do:

```
sudo python setup.py install
```

**Note:** As with any Python installation, this will install the modules in your system Python site-packages directory (which is why you need `sudo`). Many of us prefer to install development packages in a local directory so as to leave the system python alone. This is merely a preference, nothing will go wrong if you install using the `sudo` method.

If you have Python 2.6 or later, you might want to do a user install

```
python setup.py install --user
```

To install nipy in some other local directory, use the `-prefix` option. For example, if you created a local directory in your home directory, you would install nipy like this:

```
python setup.py install --prefix=$HOME/local
```

**Installing useful data files**

See data-files for some instructions on installing data packages.
CHAPTER THREE

GEOGRAPHY OF THE SCIPY WORLD

in which we briefly describe the various components you are likely to come across when writing scientific python software in general, and NIPY code in particular.

3.1 Numpy

NumPy is the basic Python array-manipulation package. It allows you to create, slice and manipulate N-D arrays at near C speed. It also has basic arithmetical and mathematical functions (such as sum, mean, and log, exp, sin, cos), matrix multiplication (numpy.dot), Fourier transforms (numpy.fft) and basic linear algebra numpy.linalg.

3.2 SciPy

Scipy is a large umbrella project that builds on Numpy (and depends on it). It includes a variety of high level science and engineering modules together as a single package. There are extended modules for linear algebra (including wrappers to BLAS and LAPACK), optimization, integration, sparse matrices, special functions, FFTs, signal and image processing, genetic algorithms, ODE solvers, and others.

3.3 Matplotlib

Matplotlib is a 2D plotting package that depends on NumPy. It has a simple matlab-like plotting syntax that makes it relatively easy to create good-looking plots, histograms and images with a small amount of code. As well as this simplified Matlab-like syntax, There is also a more powerful and flexible object-oriented interface.

3.4 Ipython

Ipython is an interactive shell for python that has various features of the interactive shell of Matlab, Mathematica and R. It works particularly well with Matplotlib, but is also an essential tool for interactive code development and code exploration. It contains libraries for creainteracting with parallel jobs on clusters or over several CPU cores in a fairly transparent way.
3.5 Cython

Cython is a development language that allows you to write a combination of Python and C-like syntax to generate Python extensions. It is especially good for linking C libraries to Python in a readable way. It is also an excellent choice for optimization of Python code, because it allows you to drop down to C or C-like code at your bottlenecks without losing much of the readability of Python.

3.6 Mayavi

Mayavi is a high-level python interface to the VTK plotting libraries.
4.1 Basic Data IO

Accessing images using nipy:

While Nifti is the primary file format Analyze images (with associated .mat file), and MINC files can also be read.

4.1.1 Load Image from File

Get a filename for an example file. anatfile gives a filename for a small testing image in the nipy distribution:

```python
>>> from nipy.testing import anatfile
```

Load the file from disk:

```python
>>> from nipy import load_image
>>> myimg = load_image(anatfile)
>>> myimg.shape
(33, 41, 25)
>>> myimg.affine
array([[ -2.,  0.,  0., 32.],
       [  0.,  2.,  0., -40.],
       [  0.,  0.,  2., -16.],
       [  0.,  0.,  0.,  1.]])
```

4.1.2 Access Data into an Array

This allows the user to access data as a numpy array.

```python
>>> mydata = myimg.get_data()
>>> mydata.shape
(33, 41, 25)
>>> mydata.ndim
3
```

4.1.3 Save image to a File

```python
>>> from nipy import save_image
>>> newimg = save_image(myimg, 'newmyfile.nii')
```
4.1.4 Create Image from an Array

This will have a generic affine-type CoordinateMap with unit voxel sizes.

```python
>>> import numpy as np
>>> from nipy.core.api import Image, vox2mni

>>> rawarray = np.zeros((43,128,128))
>>> arr_img = Image(rawarray, vox2mni(np.eye(4)))
>>> arr_img.shape
(43, 128, 128)
```

4.1.5 Coordinate map

Images have a Coordinate Map.

The Coordinate Map contains information defining the input (domain) and output (range) Coordinate Systems of the image, and the mapping between the two Coordinate systems. The input coordinate system is the voxel coordinate system, and the output coordinate system is the world coordinate system.

```python
>>> newimg.coordmap
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k'), name='voxels', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('aligned-x=L->R', 'aligned-y=P->A', 'aligned-z=I->S'), name='aligned', coord_dtype=float64),
    affine=array([[ -2.,  0.,  0.,  32.],
                  [  0.,  2.,  0., -40.],
                  [  0.,  0.,  2., -16.],
                  [  0.,  0.,  0.,   1.]]))
```

See Basics of the Coordinate Map for more detail.

4.2 Basics of the Coordinate Map

When you load an image it will have an associated Coordinate Map

Coordinate Map

The Coordinate Map contains information defining the input (domain) and output (range) Coordinate Systems of the image, and the mapping between the two Coordinate systems.

The input or domain in an image are voxel coordinates in the image array. The output or range are the millimetre coordinates in some space, that correspond to the input (voxel) coordinates.

```python
>>> import nipy

Get a filename for an example file:

```
A Coordinate Map has a mapping from the *input* Coordinate System to the *output* Coordinate System.

Here we can see we have a voxel to millimeter mapping from the voxel space (i,j,k) to the millimeter space (x,y,z).

We can also get the name of the respective Coordinate Systems that our Coordinate Map maps between.

A Coordinate Map is two Coordinate Systems with a mapping between them. Formally the mapping is a function that takes points from the input Coordinate System and returns points from the output Coordinate System. This is the same as saying that the mapping takes points in the mapping function *domain* and transforms them to points in the mapping function *range*.

Often this is simple as applying an Affine transform. In that case the Coordinate System may well have an affine property which returns the affine matrix corresponding to the transform.

If you call the Coordinate Map you will apply the mapping function between the two Coordinate Systems. In this case from (i,j,k) to (x,y,z):

\[
\begin{bmatrix}
-2. & 0. & 0. & 32. \\
0. & 2. & 0. & -40. \\
0. & 0. & 2. & -16. \\
0. & 0. & 0. & 1.
\end{bmatrix}
\]

If you call the Coordinate Map you will apply the mapping function between the two Coordinate Systems. In this case from (i,j,k) to (x,y,z):

\[
\begin{bmatrix}
1. \\
2. \\
3.
\end{bmatrix}
\]

We can see how this works if we just apply the affine ourselves using dot product.

Note: Notice the affine is using homogeneous coordinates so we need to add a 1 to our input. (And note how a direct call to the coordinate map does this work for you)

```python
>>> coordmap.affine
array([[ -2., 0., 0., 32.],
       [ 0., 2., 0., -40.],
       [ 0., 0., 2., -16.],
       [ 0., 0., 0., 1.]])
```

```python
>>> import numpy as np
>>> np.dot(coordmap.affine, np.transpose([1,2,3,1]))
array([ 30., -36., -10., 1.])
```

Note: The answer is the same as above (except for the added 1)
4.2.1 Use of the Coordinate Map for spatial normalization

The Coordinate Map can be used to describe the transformations needed to perform spatial normalization. Suppose we have an anatomical Image from one subject `subject_img` and we want to create an Image in a standard space like Tailarach space. An affine registration algorithm will produce a 4-by-4 matrix representing the affine transformation, \( T \), that takes a point in the subject’s coordinates `subject_world` to a point in Tailarach space `tailarach_world`. The subject’s Image has its own Coordinate Map, `subject_cmap` and there is a Coordinate Map for Tailarach space which we will call `tailarach_cmap`.

Having found the transformation matrix \( T \), the next step in spatial normalization is usually to resample the array of `subject_img` so that it has the same shape as some atlas `atlas_img`. Note that because it is an atlas Image, `tailarach_cmap=atlas_img.coordmap`.

A resampling algorithm uses an interpolator which needs to know which voxel of `subject_img` corresponds to which voxel of `atlas_img`. This is therefore a function from `atlas_voxel` to `subject_voxel`.

This function, paired with the information that it is a map from atlas-voxel to subject-voxel is another example of a Coordinate Map. The code to do this might look something like the following:

```python
>>> from nipy.testing import anatfile, funcfile
>>> from nipy.algorithms.registration import HistogramRegistration
>>> from nipy.algorithms.kernel_smooth import LinearFilter

We’ll make a smoothed version of the anatomical example image, and pretend it’s the template

```python
>>> smoother = LinearFilter(anat_img.coordmap, anat_img.shape)
>>> atlas_im = smoother.smooth(anat_img)
>>> subject_im = anat_img
```

We do an affine registration between the two.

```python
>>> reggie = HistogramRegistration(subject_im, atlas_im)
>>> aff = reggie.optimize('affine').as_affine()

Initial guess...
```

Now we make a coordmap with this transformation

```python
>>> from nipy.core.api import AffineTransform
>>> subject_cmap = subject_im.coordmap
>>> talairach_cmap = atlas_im.coordmap
>>> subject_world_to_talairach_world = AffineTransform(
...    subject_cmap.function_range,
...    talairach_cmap.function_range,
...    aff)
```

We resample the ‘subject’ image to the ‘atlas image’

```python
>>> from nipy.algorithms.resample import resample
>>> normalized_subject_im = resample(subject_im, talairach_cmap, subject_world_to_talairach_world, atlas_im.shape)
>>> normalized_subject_im.shape == atlas_im.shape
True
>>> normalized_subject_im.coordmap == atlas_im.coordmap
True
```
Neuroimaging in Python Documentation, Release 0.3.0

```python
>>> np.all(normalized_subject_im.affine == atlas_im.affine)
True
```

4.2.2 Mathematical definition

For a more formal mathematical description of the coordinate map, see math-coordmap.

4.3 Specifying a GLM in NiPy

In this tutorial we will discuss NiPy’s model and specification of a fMRI experiment.

This involves:

- an experimental model: a description of the experimental protocol (function of experimental time)
- a neuronal model: a model of how a particular neuron responds to the experimental protocol (function of the experimental model)
- a hemodynamic model: a model of the BOLD signal at a particular voxel, (function of the neuronal model)

4.3.1 Experimental model

We first begin by describing typically encountered fMRI designs.

- Event-related categorical design, i.e. Face vs. Object
- Block categorical design
- Continuous stimuli, i.e. a rotating checkerboard
- Events with amplitudes, i.e. non-categorical values
- Events with random amplitudes

Event-related categorical design

This design is a canonical design in fMRI used, for instance, in an experiment designed to detect regions associated to discrimination between Face and Object. This design can be graphically represented in terms of delta-function responses that are effectively events of duration 0 and infinite height.
In this example, there `Face` event types are presented at times \([0,4,8,12,16]\) and `Object` event types at times \([2,6,10,14,18]\).

More generally, given a set of event types \(V\), an event type experiment can be modeled as a sum of delta functions (point masses) at pairs of times and event types:

\[
E = \sum_{j=1}^{10} \delta(t_j, a_j).
\]

Formally, this can be thought of as realization of a marked point process, that says we observe 10 points in the space \(\mathbb{R} \times V\) where \(V\) is the set of all event types. Alternatively, we can think of the experiment as a measure \(E\) on \(\mathbb{R} \times V\)

\[
E([t_1, t_2] \times A) = \int_{t_1}^{t_2} \int_A dE(v, t)
\]

This intensity measure determines, in words, “the amount of stimulus within \(A\) delivered in the interval \([t_1, t_2]\)”. In this categorical design, stimuli \(a_j\) are delivered as point masses at the times \(t_j\).

Practically speaking, we can read this as saying that our experiment has 10 events, occurring at times \(t_1, \ldots, t_{10}\) with event types \(a_1, \ldots, a_{10} \in V\).

Typically, as in our `Face` vs `Object` example, the events occur in groups, say odd events are labelled \(a\), even ones \(b\). We might rewrite this as

\[
E = \delta(t_1, a) + \delta(t_2, b) + \delta(t_3, a) + \cdots + \delta(t_{10}, b)
\]
This type of experiment can be represented by two counting processes, i.e. measures on $\mathbb{R}$, $(E_a, E_b)$ defined as

$$E_a(t) = \sum_{t_j, j \text{ odd}} 1_{(-\infty, t_j]}(t)$$

$$= E(((-\infty, t], \{a\})$$

$$E_b(t) = \sum_{t_j, j \text{ even}} 1_{(-\infty, t_j]}(t)$$

$$= E(((-\infty, t], \{b\})$$

**Counting processes vs. intensities**

Though the experiment above can be represented in terms of the pair $(E_a(t), E_b(t))$, it is more common in neuroimaging applications to work with instantaneous intensities rather than cumulative intensities.

$$e_a(t) = \frac{\partial}{\partial t} E_a(t)$$

$$e_b(t) = \frac{\partial}{\partial t} E_b(t)$$

For the time being, we will stick with cumulative intensities because it unifies the designs above. When we turn to the neuronal model below, we will return to the intensity model.

**Block categorical design**

For block designs of the *Face vs. Object* type, we might also allow event durations, meaning that we show the subjects a *Face* for a period of, say, 0.5 seconds. We might represent this experiment graphically as follows,
and the intensity measure for the experiment could be expressed in terms of

\[ E_a(t) = E((\infty, t], \{a\}) = \sum_{t_j, j \text{ odd}} \frac{1}{0.5} \int_{t_j}^{\min(t_j + 0.5, t)} ds \]

\[ E_b(t) = E((\infty, t], \{b\}) = \sum_{t_j, j \text{ even}} \frac{1}{0.5} \int_{t_j}^{\min(t_j + 0.5, t)} ds \]

The normalization chosen above ensures that each event has integral 1, that is a total of 1 “stimulus unit” is presented for each 0.5 second block. This may or may not be desirable, and could easily be changed.

**Continuous stimuli**

Some experiments do not fit well into this “event-type” paradigm but are, rather, more continuous in nature. For instance, a rotating checkerboard, for which orientation, contrast, are functions of experiment time \( t \). This experiment can be represented in terms of a state vector \( (O(t), C(t)) \). In this example we have set

```python
import numpy as np

import numpy as np

# Define time range
import numpy as np

t = np.linspace(0, 10, 1000)

# Define intensities

o = np.sin(2 * np.pi * (t + 2)) * np.exp(-t/10)
c = np.sin(2 * np.pi * (t + 0.2) / 4) * np.exp(-t/12)
```
The cumulative intensity measure for such an experiment might look like

\[ E([t_1, t_2], A) = \int_{t_1}^{t_2} \left( \int_A dC \, dO \right) \, dt. \]

In words, this reads as \( E([t_1, t_2], A) \) is the amount of time in the interval \([t_1, t_2]\) for which the state vector \((O(t), C(t))\) was in the region \(A\).

**Events with amplitudes**

Another (event-related) experimental paradigm is one in which the event types have amplitudes, perhaps in a pain experiment with a heat stimulus, we might consider the temperature an amplitude. These amplitudes could be multi-valued. We might represent this parametric design mathematically as

\[ E = \sum_{j=1}^{10} \delta(t_j, a_j), \]

which is virtually identical to our description of the *Face vs. Object* experiment in *face-object* though the values \(a_j\) are floats rather than labels. Graphically, this experiment might be represented as in this figure below.
Events with random amplitudes

Another possible approach to specifying an experiment might be to deliver a randomly generated stimulus, say, uniformly distributed on some interval, at a set of prespecified event times.

We might represent this graphically as in the following figure.
Of course, the stimuli need not be randomly distributed over some interval, they could have fairly arbitrary distributions. Or, in the *Face vs Object* scenario, we could randomly present one of the two types and the distribution at a particular event time $t_j$ would be represented by a probability $P_j$.

The cumulative intensity model for such an experiment might be

$$E([t_1, t_2], A) = \sum_j 1_{[t_1, t_2]}(t_j) \int_A P_j(da)$$

If the times were not prespecified but were themselves random, say uniform over intervals $[u_j, v_j]$, we might modify the cumulative intensity to be

$$E([t_1, t_2], A) = \sum_j \int_{\max(u_j, t_1)}^{\min(v_j, t_2)} \int_A P_j(da) \, dt$$

**4.3. Specifying a GLM in NiPy**
4.4 Neuronal model

The neuronal model is a model of the activity as a function of \( t \) at a neuron \( x \) given the experimental model \( E \). It is most commonly expressed as some linear function of the experiment \( E \). As with the experimental model, we prefer to start off by working with the cumulative neuronal activity, a measure on \( \mathbb{R} \), though, ultimately we will work with the intensities in intensity.

Typically, the neuronal model with an experiment model \( E \) has the form

\[
N([t_1,t_2]) = \int_{t_1}^{t_2} \int_V f(v,t) \, dE(v,t)
\]

Unlike the experimental model, which can look somewhat abstract, the neuronal model can be directly modeled. For example, take the standard *Face vs. Object* model *face-object*, in which case \( V = \{a,b\} \) and we can set

\[
f(v,t) = \begin{cases} 
    \beta_a & v = a \\
    \beta_b & v = b 
\end{cases}
\]

Thus, the cumulative neuronal model can be expressed as

```python
from sympy import Symbol, Heaviside
t = Symbol('t')
ta = [0,4,8,12,16]
```
tb = [2, 6, 10, 14, 18]
ba = Symbol('ba')
bb = Symbol('bb')
fa = sum([Heaviside(_t-t) for _t in ta]) * ba
fb = sum([Heaviside(_t-t) for _t in tb]) * bb
N = fa + fb

Or, graphically, if we set $\beta_a = 1$ and $\beta_b = -2$, as

In the block design, we might have the same form for the neuronal model (i.e. the same \( f \) above), but the different experimental model \( E \) yields

```python
from sympy import Symbol, Piecewise
ta = [0, 4, 8, 12, 16]; tb = [2, 6, 10, 14, 18]
ba = Symbol('ba')
bb = Symbol('bb')
fa = sum([Piecewise((0, (t<_t)), ((t-_t)/0.5, (t<_t+0.5)), (1, (t >= _t+0.5))) for _t in ta])*ba
fb = sum([Piecewise((0, (t<_t)), ((t-_t)/0.5, (t<_t+0.5)), (1, (t >= _t+0.5))) for _t in tb])*bb
N = fa + fb
```

Or, graphically, if we set $\beta_a = 1$ and $\beta_b = -2$, as

4.4. Neuronal model
The function $f$ above can be expressed as

$$f(v, t) = \beta_a 1_{\{a\}}(v) + \beta_b 1_{\{b\}}(v) = \beta_a f_a(v, t) + \beta_b f_b(v, t)$$

Hence, our typical neuronal model can be expressed as a sum

$$N([t_1, t_2]) = \sum_{i} \beta_i \int_{t_1}^{t_2} \int_{V} f_i(v, t) \, dE(v, t)$$

for arbitrary functions $\hat{N}_f$. Above, $\hat{N}_f$ represents the stimulus contributed to $N$ from the function $f_i$. In the Face vs. Object example face-object, these cumulative intensities are related to the more common of neuronal model of intensities in terms of delta functions

$$\frac{\partial}{\partial t} \hat{N}_f(t) = \beta_a \sum_{i; i \text{ odd}} \delta_{t_i}(t)$$

```python
from sympy import Symbol, Heaviside
ta = [0, 4, 8, 12, 16]
t = Symbol('t')
ba = Symbol('ba')
fa = sum([Heaviside(t-_t) for _t in ta]) * ba
print(fa.diff(t))
```
4.4.1 Convolution

In our continuous example above, with a periodic orientation and contrast, we might take

\[ f_O(t, (o, c)) = o \]
\[ f_O(t, (o, c)) = c \]

yielding a neuronal model

\[ N([t_1, t_2]) = \beta_O O(t) + \beta_C C(t) \]

We might also want to allow a delay in the neuronal model

\[ N^{\text{delay}}([t_1, t_2]) = \beta_O O(t - \tau_O) + \beta_C C(t - \tau_C). \]

This delay can be represented mathematically in terms of convolution (of measures)

\[ N^{\text{delay}}([t_1, t_2]) = \left( \hat{N}_{f_O} \ast \delta_{-\tau_O} \right) ([t_1, t_2]) + \left( \hat{N}_{f_C} \ast \delta_{-\tau_C} \right) ([t_1, t_2]) \]

Another model that uses convolution is the **Face vs. Object** one in which the neuronal signal is attenuated with an exponential decay at time scale \( \tau \)

\[ D([t_1, t_2]) = \int_{\max(t_1, 0)}^{t_2} \tau e^{-\tau t} \, dt \]
yielding

\[ N^{\text{decay}}([t_1, t_2]) = (N \ast D)[t_1, t_2] \]

### 4.5 Events with amplitudes

We described a model above *event-amplitude* with events that each have a continuous value \( a \) attached to them. In terms of a neuronal model, it seems reasonable to suppose that the (cumulative) neuronal activity is related to some function, perhaps expressed as a polynomial \( h(a) = \sum_j \beta_j a^j \) yielding a neuronal model

\[ N([t_1, t_2]) = \sum_j \beta_j \tilde{N}_a, ([t_1, t_2]) \]

### 4.5.1 Hemodynamic model

The hemodynamic model is a model for the BOLD signal, expressed as some function of the neuronal model. The most common hemodynamic model is just the convolution of the neuronal model with some hemodynamic response function, \( HRF \)

\[
HRF(\left[ -\infty, t \right]) = \int_{-\infty}^{t} h_{\text{can}}(s) \, ds \\
H([t_1, t_2]) = (N \ast HRF)[t_1, t_2]
\]

The canonical one is a difference of two Gamma densities
4.5.2 Intensities

Hemodynamic models are, as mentioned above, most commonly expressed in terms of instantaneous intensities rather than cumulative intensities. Define

\[ n(t) = \frac{\partial}{\partial t} N((-\infty, t]). \]

The simple model above can then be written as

\[ h(t) = \frac{\partial}{\partial t}(N \ast HRF)(t) = \int_{-\infty}^{\infty} n(t - s)h_{can}(s) \, ds. \]

In the *Face vs. Object* experiment, the integrals above can be evaluated explicitly because \( n(t) \) is a sum of delta functions

\[ n(t) = \beta_a \sum_{t_i: \text{odd}} \delta(t - t_i) + \beta_b \sum_{t_i: \text{even}} \delta(t - t_i) \]

In this experiment we may want to allow different hemodynamic response functions within each group, say \( h_a \) within group \( a \) and \( h_b \) within group \( b \). This yields a hemodynamic model

\[ h(t) = \beta_a \sum_{t_i: \text{odd}} h_a(t - t_i) + \beta_b \sum_{t_i: \text{even}} h_b(t - t_i) \]

```
from nipy.modalities.fmri import hrf
ta = [0, 4, 8, 12, 16]; tb = [2, 6, 10, 14, 18]
ba = 1; bb = -2
na = ba * sum([hrf.glover(hrf.T - t) for t in ta])
nb = bb * sum([hrf.afni(hrf.T - t) for t in tb])
n = na + nb
```
Applying the simple model to the events with amplitude model and the canonical HRF yields a hemodynamic model

\[ h(t) = \sum_{i,j} \beta_j a_i^2 h_{can}(t - t_i) \]

```python
import numpy as np
from nipy.modalities.fmri.utils import events, Symbol

a = Symbol('a')
b = np.linspace(0, 50, 6)
amp = b*(-1,1)*3
d = events(b, amplitudes=amp, g=a+0.5*a**2, f=hrf.glover)
```
4.5.3 Derivative information

In cases where the neuronal model has more than one derivative, such as the continuous stimuli example, we might model the hemodynamic response using the higher derivatives as well. For example

\[ h(t) = \beta_{O,0} \hat{n}_{f0}(t) + \beta_{O,1} \frac{\partial}{\partial t} \hat{n}_{f0}(t) + \beta_{C,0} \hat{n}_{fc}(t) + \beta_{C,1} \frac{\partial}{\partial t} \hat{n}_{fc}(t) \]

where

\[ \hat{n}_f(t) = \frac{\partial}{\partial t} \tilde{N}_f((-\infty, t]) \]
\[ = \frac{\partial}{\partial t} \left( \int_{-\infty}^{t} \int_{V} f(v, t) \, dE(v, t) \right) \]

4.6 Design matrix

In a typical GLM analysis, we will compare the observed BOLD signal \( B(t) \) at some fixed voxel \( x \), observed at time points \( \{s_1, \ldots, s_n\} \), to a hemodynamic response model. For instance, in the Face vs. Object model, using the canonical HRF

\[ B(t) = \beta_a \sum_{t_i: \text{odd}} h_{can}(t - t_i) + \beta_b \sum_{t_i: \text{even}} h_{can}(t - t_i) + \epsilon(t) \]
where $\epsilon(t)$ is the correlated noise in the BOLD data.

Because the BOLD is modeled as linear in $(\beta_a, \beta_b)$ this fits into a multiple linear regression model setting, typically written as

$$Y_{n \times 1} = X_{n \times p}\beta_{p \times 1} + \epsilon_{n \times 1}$$

In order to fit the regression model, we must find the matrix $X$. This is just the derivative of the model of the mean of $B$ with respect to the parameters to be estimated. Setting $(\beta_1, \beta_2) = (\beta_a, \beta_b)$

$$X_{ij} = \frac{\partial}{\partial\beta_j} \left( \beta_1 \sum_{t_k: k \text{ odd}} h_{can}(s_i - t_k) + \beta_b \sum_{t_k: k \text{ even}} h_{can}(s_i - t_k) \right)$$

### 4.6.1 Drift

We sometimes include a natural spline model of the drift here.

This changes the design matrix by adding more columns, one for each function in our model of the drift. In general, starting from some model of the mean the design matrix is the derivative of the model of the mean, differentiated with respect to all parameters to be estimated (in some fixed order).

### 4.6.2 Nonlinear example

The delayed continuous stimuli example above is an example of a nonlinear function of the mean that is nonlinear in some parameters, $(\tau_O, \tau_C)$.

### 4.7 Formula objects

This experience of building the model can often be simplified, using what is known in R as formula objects. NiPy has implemented a formula object that is similar to R’s, but differs in some important respects. See nipy.algorithms.statistics.formula.
AFNI  AFNI is a functional imaging analysis package. It is funded by the NIMH, based in Bethesda, Maryland, and directed by Robert Cox. Like FSL, it is written in C, and it's very common to use shell scripting of AFNI command line utilities to automate analyses. Users often describe liking AFNI’s scriptability, and image visualization. It uses the GPL license.

BSD  Berkeley software distribution license. The BSD license is permissive, in that it allows you to modify and use the code without requiring that you use the same license. It allows you to distribute closed-source binaries.

BOLD  Contrast that is blood oxygen level dependent. When a brain area becomes active, blood flow increases to that area. It turns out that, with the blood flow increase, there is a change in the relative concentrations of oxygenated and deoxygenated hemoglobin. Oxy- and deoxy- hemoglobin have different magnetic properties. This in turn leads to a change in MRI signal that can be detected by collecting suitably sensitive MRI images at regular short intervals during the blood flow chance. See the wikipedia FMRI article for more detail.

BrainVisa  BrainVISA is a sister project to NIPY. It also uses Python, and provides a carefully designed framework and automatic GUI for defining imaging processing workflows. It has tools to integrate command line and other utilities into these workflows. Its particular strength is anatomical image processing but it also supports FMRI and other imaging modalities. BrainVISA is based in NeuroSpin, outside Paris.

DTI  Diffusion tensor imaging. DTI is rather poorly named, because it is a model of the diffusion signal, and an analysis method, rather than an imaging method. The simplest and most common diffusion tensor model assumes that diffusion direction and velocity at every voxel can be modeled by a single tensor - that is, by an ellipse of regular shape, fully described by the length and orientation of its three orthogonal axes. This model can easily fail in fairly common situations, such as white-matter fiber track crossings.

DWI  Diffusion-weighted imaging. DWI is the general term for MRI imaging designed to image diffusion processes. Sometimes researchers use DTI to have the same meaning, but DTI is a common DWI signal model and analysis method.

EEGlab  The most widely-used open-source package for analyzing electrophysiological data. EEGlab is written in matlab and uses a GPL license.

FMRI  Functional magnetic resonance imaging! It refers to MRI image acquisitions and analysis designed to look at brain function rather than structure. Most people use FMRI to refer to BOLD imaging in particular. See the wikipedia FMRI article for more detail.

FSL  FSL is the FMRIB software library, written by the FMRIB analysis group, and directed by Steve Smith. Like AFNI, it is a large collection of C / C++ command line utilities that can be scripted with a custom GUI / batch system, or using shell scripting. Its particular strength is analysis of DWI data, and ICA functional data analysis, although it has strong tools for the standard SPM approach to FMRI. It is free for academic use, and open-source, but not free for commercial use.

GPL  The GPL is the GNU general public license. It is one of the most commonly-used open-source sofware licenses. The distinctive feature of the GPL license is that it requires that any code derived from GPL code also uses a GPL
ICA  Independent component analysis is a multivariate technique related to PCA, to estimate independent components of signal from multiple sensors. In functional imaging, this usually means detecting underlying spatial and temporal components within the brain, where the brain voxels can be considered to be different sensors of the signal. See the wikipedia ICA page.

LGPL  The lesser GNU public license. LGPL differs from the GPL in that you can link to LGPL code from non-LGPL code without having to adopt a GPL-compatible license. However, if you modify the code (create a “derivative work”), that modification has to be released under the LGPL. See wikipedia LGPL for more discussion.

Matlab  Matlab began as a high-level programming language for working with matrices. Over time it has expanded to become a fairly general-purpose language. See also: http://en.wikipedia.org/wiki/MATLAB. It has good numerical algorithms, 2D graphics, and documentation. There are several large neuroscience software projects written in matlab, including SPM software, and EEGLab.

PCA  Principal component analysis is a multivariate technique to determine orthogonal components across multiple sources (or sensors). See ICA and the wikipedia PCA page.

PET  Positron emission tomography is a method of detecting the spatial distributions of certain radiolabeled compounds - usually in the brain. The scanner detectors pick up the spatial distribution of emitted radiation from within the body. From this pattern, it is possible to reconstruct the distribution of radioactivity in the body, using techniques such as filtered back projection. PET was the first mainstream technique used for detecting regional changes in blood-flow as an index of which brain areas were active when the subject is doing various tasks, or at rest. These studies nearly all used water activation PET. See the wikipedia PET entry.

SPM  SPM (statistical parametric mapping) refers either to the SPM approach to analysis or the SPM software package.

SPM approach  Statistical parametric mapping is a way of analyzing data, that involves creating an image (the map) containing statistics, and then doing tests on this statistic image. For example, we often create a t statistic image where each voxel contains a t statistic value for the time-series from that voxel. The SPM software package implements this approach - as do several others, including FSL and AFNI.

SPM software  SPM (statistical parametric mapping) is the name of the matlab based package written by John Ashburner, Karl Friston and others at the Functional Imaging Laboratory in London. More people use the SPM package to analyze FMRI and PET data than any other. It has good lab and community support, and the matlab source code is available under the GPL license.

VoxBo  Quoting from the Voxbo webpage - “VoxBo is a software package for the processing, analysis, and display of data from functional neuroimaging experiments”. Like SPM, FSL and AFNI, VoxBo provides algorithms for a full FMRI analysis, including statistics. It also provides software for lesion-symptom analysis, and has a parallel scripting engine. VoxBo has a GPL license. Dan Kimberg leads development.

voxel  Voxels are volumetric pixels - that is, they are values in a regular grid in three dimensional space - see http://en.wikipedia.org/wiki/Voxel

water activation PET  A PET technique to detect regional changes in blood flow. Before each scan, we inject the subject with radiolabeled water. The radiolabeled water reaches the arterial blood, and then distributes (to some extent) in the brain. The concentration of radioactive water increases in brain areas with higher blood flow. Thus, the image of estimated counts in the brain has an intensity that is influenced by blood flow. This use has been almost completely replaced by the less invasive BOLD FMRI technique.
Part II

NeuroSpin tools
The package `nipy.labs` hosts some tools that were originally developed at NeuroSpin, France. The list below also includes routines for estimating the empirical null, moved from `nipy.labs` to `nipy.algorithms.statistics`.
The module `nipy.labs.utils.mask` contains utilities to extract brain masks from fMRI data:

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>compute_mask(mean_volume[, ...])</code></td>
<td>Compute a mask file from fMRI data in 3D or 4D ndarrays.</td>
</tr>
<tr>
<td><code>compute_mask_files(input_filename[, ...])</code></td>
<td>Compute a mask file from fMRI nifti file(s)</td>
</tr>
<tr>
<td><code>compute_mask_sessions(session_images[, m, ...])</code></td>
<td>Compute a common mask for several sessions of fMRI data.</td>
</tr>
</tbody>
</table>

### 6.1 nipy.labs.utils.mask.compute_mask

Compute a mask file from fMRI data in 3D or 4D ndarrays.

**Parameters**
- `mean_volume`: 3D ndarray
  - mean EPI image, used to compute the threshold for the mask.
- `reference_volume`: 3D ndarray, optional
  - reference volume used to compute the mask. If none is give, the mean volume is used.
- `m`: float, optional
  - lower fraction of the histogram to be discarded.
- `M`: float, optional
  - upper fraction of the histogram to be discarded.
- `cc`: boolean, optional
  - if `cc` is True, only the largest connect component is kept.
- `opening`: int, optional
  - if opening is larger than 0, an morphological opening is performed, to keep only large structures. This step is useful to remove parts of the skull that might have been included.
- `exclude_zeros`: boolean, optional
  - Consider zeros as missing values for the computation of the threshold. This option is useful if the images have been resliced with a large padding of zeros.
Returns mask: 3D boolean ndarray

The brain mask

6.2 nipy.labs.utils.mask.compute_mask_files

nipy.labs.utils.mask.compute_mask_files(input_filename, output_filename=None, return_mean=False, m=0.2, M=0.9, cc=1, exclude_zeros=False, opening=2)

Compute a mask file from fMRI nifti file(s)

Compute and write the mask of an image based on the grey level. This is based on an heuristic proposed by T.Nichols: find the least dense point of the histogram, between fractions m and M of the total image histogram. In case of failure, it is usually advisable to increase m.

Parameters input_filename: string

nifti filename (4D) or list of filenames (3D).

output_filename: string or None, optional

path to save the output nifti image (if not None).

return_mean: boolean, optional

if True, and output_filename is None, return the mean image also, as a 3D array (2nd return argument).

m: float, optional

lower fraction of the histogram to be discarded.

M: float, optional

upper fraction of the histogram to be discarded.

cc: boolean, optional

if cc is True, only the largest connect component is kept.

exclude_zeros: boolean, optional

Consider zeros as missing values for the computation of the threshold. This option is useful if the images have been resliced with a large padding of zeros.

opening: int, optional

Size of the morphological opening performed as post-processing

Returns mask: 3D boolean array

The brain mask

mean_image: 3d ndarray, optional

The mean of all the images used to estimate the mask. Only provided if return_mean is True.
6.3  nipy.labs.util.mask.compute_mask_sessions

nipy.labs.util.mask.compute_mask_sessions(session_images, m=0.2, M=0.9, cc=1,
threshold=0.5, exclude_zeros=False, return_mean=False, opening=2)

Compute a common mask for several sessions of fMRI data.

Uses the mask-finding algorithms to extract masks for each session, and then keep only the main
connected component of the a given fraction of the intersection of all the masks.

Parameters  session_images : list of (list of strings) or nipy image objects

A list of images/list of nifti filenames. Each inner list/image represents a session.

m : float, optional
    lower fraction of the histogram to be discarded.

M: float, optional :
    upper fraction of the histogram to be discarded.

cc: boolean, optional :
    if cc is True, only the largest connect component is kept.

threshold : float, optional
    the inter-session threshold: the fraction of the total number of session in for which a
    voxel must be in the mask to be kept in the common mask. threshold=1 corresponds to
    keeping the intersection of all masks, whereas threshold=0 is the union of all masks.

exclude_zeros: boolean, optional :
    Consider zeros as missing values for the computation of the threshold. This option is
    useful if the images have been resliced with a large padding of zeros.

return_mean: boolean, optional :
    if return_mean is True, the mean image accross subjects is returned.

opening: int, optional, :
    size of the morphological opening

Returns  mask : 3D boolean ndarray

The brain mask

mean : 3D float array

The mean image

The compute_mask_files() and compute_mask_sessions() functions work with Nifti files rather than
numpy ndarrays. This is convenient to reduce memory pressure when working with long time series, as there is no
need to store the whole series in memory.
CHAPTER
SEVEN

EMPIRICAL NULL

The `nipy.algorithms.statistics.empirical_pvalue` module contains a class that fits a Gaussian model to the central part of an histogram, following Schwartzman et al, 2009. This is typically necessary to estimate a FDR when one is not certain that the data behaves as a standard normal under $H_0$.

The `NormalEmpiricalNull` class learns its null distribution on the data provided at initialisation. Two different methods can be used to set a threshold from the null distribution: the `NormalEmpiricalNull.threshold()` method returns the threshold for a given false discovery rate, and thus accounts for multiple comparisons with the given dataset; the `NormalEmpiricalNull.uncorrected_threshold()` returns the threshold for a given uncorrected $p$-value, and as such does not account for multiple comparisons.

7.1 Example

If we use the empirical normal null estimator on a two Gaussian mixture distribution, with a central Gaussian, and a wide one, it uses the central distribution as a null hypothesis, and returns the threshold following which the data can be claimed to belong to the wide Gaussian:

```python
# emacs: -*- mode: python; py-indent-offset: 4; indent-tabs-mode: nil -*-
# vi: set ft=python sts=4 ts=4 sw=4 et:
import numpy as np

from nipy.algorithms.statistics.empirical_pvalue import NormalEmpiricalNull

x = np.c_[np.random.normal(size=1e4),
         np.random.normal(scale=4, size=1e4)]

enn = NormalEmpiricalNull(x)
enn.threshold(verbos=True)
```
The threshold evaluated with the `NormalEmpiricalNull.threshold()` method is around 2.8 (using the default p-value of 0.05). The `NormalEmpiricalNull.uncorrected_threshold()` returns, for the same p-value, a threshold of 1.9. It is necessary to use a higher p-value with uncorrected comparisons.

### 7.2 Class documentation

**class** `nipy.algorithms.statistics.empirical_pvalue.NormalEmpiricalNull(x)`

Class to compute the empirical null normal fit to the data.

The data which is used to estimate the FDR, assuming a Gaussian null from Schwartzmann et al., NeuroImage 44 (2009) 71–82

**Methods**

- `fdr(theta)` Given a threshold theta, find the estimated FDR
- `fdrcurve()` Returns the FDR associated with any point of self.x
- `learn([left, right])` Estimate the proportion, mean and variance of a Gaussian distribution
- `plot([efp, alpha, bar, mpaxes])` Plot the histogram of x
- `threshold([alpha, verbose])` Compute the threshold corresponding to an alpha-level FDR for x
- `uncorrected_threshold([alpha, verbose])` Compute the threshold corresponding to a specificity alpha for x
__init__(x)
Initialize an empirical null normal object.

Parameters x : 1D ndarray
The data used to estimate the empirical null.

fdr(theta)
Given a threshold theta, find the estimated FDR

Parameters theta : float or array of shape (n_samples)
values to test

Returns afp : value of array of shape(n)

fdrcurve()
Returns the FDR associated with any point of self.x

learn(left=0.2, right=0.8)
Estimate the proportion, mean and variance of a Gaussian distribution for a fraction of the data

Parameters left: float, optional :
Left cut parameter to prevent fitting non-gaussian data

right: float, optional :
Right cut parameter to prevent fitting non-gaussian data

Notes
This method stores the following attributes:
•mu = mu
•p0 = min(1, np.exp(lp0))
•sqsigma: standard deviation of the estimated normal distribution
•sigma: np.sqrt(sqsigma) : variance of the estimated normal distribution

plot(efp=None, alpha=0.05, bar=1, mpaxes=None)
Plot the histogram of x

Parameters efp : float, optional
The empirical FDR (corresponding to x) if efp==None, the false positive rate threshold plot is not drawn.

alpha : float, optional
The chosen FDR threshold

bar=1 : bool, optional

mpaxes=None: if not None, handle to an axes where the fig:
will be drawn. Avoids creating unnecessarily new figures :

threshold(alpha=0.05, verbose=0)
Compute the threshold corresponding to an alpha-level FDR for x

Parameters alpha : float, optional
the chosen false discovery rate threshold.
verbose : boolean, optional
    the verbosity level, if True a plot is generated.

Returns  theta: float :
    the critical value associated with the provided FDR

uncorrected_threshold (alpha=0.001, verbose=0)
    Compute the threshold corresponding to a specificity alpha for x

Parameters  alpha : float, optional
    the chosen false discovery rate (FDR) threshold.

verbose : boolean, optional
    the verbosity level, if True a plot is generated.

Returns  theta: float :
    the critical value associated with the provided p-value

PLOTTING OF ACTIVATION MAPS

The module `nipy.labs.viz` provides functions to plot visualization of activation maps in a non-interactive way. 2D cuts of an activation map can be plotted and superimposed on an anatomical map using `matplotlib`. In addition, `Mayavi2` can be used to plot 3D maps, using volumetric rendering. Some emphasis is made on automatic choice of default parameters, such as cut coordinates, to give a sensible view of a map in a purely automatic way, for instance to save a summary of the output of a calculation.

**Warning:** The content of the module will change over time, as neuroimaging volumetric data structures are used instead of plain `numpy` arrays.

### 8.1 An example

```python
from nipy.labs.viz import plot_map, mni_sform, coord_transform

# First, create a fake activation map: a 3D image in MNI space with
# a large rectangle of activation around Broca Area
import numpy as np
mni_sform_inv = np.linalg.inv(mni_sform)
# Color an asymmetric rectangle around Broca area:
x, y, z = -52, 10, 22
x_map, y_map, z_map = coord_transform(x, y, z, mni_sform_inv)
map = np.zeros((182, 218, 182))
map[x_map-30:x_map+30, y_map-3:y_map+3, z_map-10:z_map+10] = 1

# We use a masked array to add transparency to the parts that we are
# not interested in:
thresholded_map = np.ma.masked_less(map, 0.5)

# And now, visualize it:
plot_map(thresholded_map, mni_sform, cut_coords=(x, y, z), vmin=0.5)
```

This creates the following image:
The same plot can be obtained fully automatically, by letting `plot_map()` find the activation threshold and the cut coordinates:

```python
plot_map(map, mni_sform, threshold='auto')
```

In this simple example, the code will easily detect the bar as activation and position the cut at the center of the bar.

### 8.2 `nipy.labs.viz` functions

#### `plot_map(map, affine[, cut_coords, anat, ...])` Plot three cuts of a given activation map (Frontal, Axial, and Lateral)

#### 8.2.1 `nipy.labs.viz_tools.activation_maps.plot_map`

`nipy.labs.viz_tools.activation_maps.plot_map(map, affine, cut_coords=None, anat=None, anat_affine=None, slicer='ortho', figure=None, axes=None, title=None, threshold=None, annotate=True, draw_cross=True, do3d=False, threshold_3d=None, view_3d=(38.5, 70.5, 300, (-2.7, -12, 9.1)), black_bg=False, **kwargs)`

Plot three cuts of a given activation map (Frontal, Axial, and Lateral)

**Parameters**

- `map` : 3D ndarray
  
  The activation map, as a 3D image.

- `affine` : 4x4 ndarray
  
  The affine matrix going from image voxel space to MNI space.

- `cut_coords` : None, or a tuple of floats
  
  The MNI coordinates of the point where the cut is performed, in MNI coordinates and order. If `slicer` is `ortho`, this should be a 3-tuple: `(x, y, z)` For `slicer` == `x`, `y`, or `z`, then these are the coordinates of each cut in the corresponding direction. If `None` is given, the cuts is calculated automatically.

- `anat` : 3D ndarray or False, optional
  
  Optional anatomical image.
The anatomical image to be used as a background. If None, the MNI152 T1 1mm template is used. If False, no anat is displayed.

**anat_affine**: 4x4 ndarray, optional

The affine matrix going from the anatomical image voxel space to MNI space. This parameter is not used when the default anatomical is used, but it is compulsory when using an explicit anatomical image.

**slicer**: {'ortho', 'x', 'y', 'z'}:

Choose the direction of the cuts. With ‘ortho’ three cuts are performed in orthogonal directions.

**figure**: integer or matplotlib figure, optional

Matplotlib figure used or its number. If None is given, a new figure is created.

**axes**: matplotlib axes or 4 tuple of float: (xmin, ymin, width, height), optional

The axes, or the coordinates, in matplotlib figure space, of the axes used to display the plot. If None, the complete figure is used.

**title**: string, optional

The title displayed on the figure.

**threshold**: a number, None, or ‘auto’

If None is given, the maps are not thresholded. If a number is given, it is used to threshold the maps: values below the threshold are plotted as transparent. If auto is given, the threshold is determined magically by analysis of the map.

**annotate**: boolean, optional:

If annotate is True, positions and left/right annotation are added to the plot.

**draw_cross**: boolean, optional:

If draw_cross is True, a cross is drawn on the plot to indicate the cut position.

**do3d**: {True, False or ‘interactive’}, optional:

If True, Mayavi is used to plot a 3D view of the map in addition to the slicing. If ‘interactive’, the 3D visualization is displayed in an additional interactive window.

**threshold_3d**: 

The threshold to use for the 3D view (if any). Defaults to the same threshold as that used for the 2D view.

**view_3d**: tuple, :

The view used to take the screenshot: azimuth, elevation, distance and focalpoint, see the docstring of mlab.view.

**black_bg**: boolean, optional:

If True, the background of the image is set to be black. If you wish to save figures with a black background, you will need to pass “facecolor='k’, edgecolor='k’” to pylab’s savefig.

**kwargs**: extra keyword arguments, optional:

Extra keyword arguments passed to pylab.imshow
8.3 3D plotting utilities

The module `nipy.labs.viz3d` can be used as helpers to represent neuroimaging volumes with Mayavi2.

```python
plot_map_3d(map, affine[, cut_coords, anat, ...])  # Plot a 3D volume rendering view of the activation, with an outline of the brain.

plot_anat_3d([anat, anat_affine, scale, ...])  # 3D anatomical display
```

8.3.1 nipy.labs.viz_tools.maps_3d.plot_map_3d

```python
nipy.labs.viz_tools.maps_3d.plot_map_3d(map, affine, cut_coords=None, anat=None, anat_affine=None, threshold=None, offscreen=False, vmin=None, vmax=None, cmap=None, view=(38.5, 70.5, 300, (-2.7, -12, 9.1)))
```

Plot a 3D volume rendering view of the activation, with an outline of the brain.

**Parameters**

- `map`: 3D ndarray
  - The activation map, as a 3D image.
- `affine`: 4x4 ndarray
  - The affine matrix going from image voxel space to MNI space.
- `cut_coords`: 3-tuple of floats, optional
  - The MNI coordinates of a 3D cursor to indicate a feature or a cut, in MNI coordinates and order.
- `anat`: 3D ndarray, optional
  - The anatomical image to be used as a background. If None, the MNI152 T1 1mm template is used. If False, no anatomical image is used.
- `anat_affine`: 4x4 ndarray, optional
  - The affine matrix going from the anatomical image voxel space to MNI space. This parameter is not used when the default anatomical is used, but it is compulsory when using an explicit anatomical image.
- `threshold`: float, optional
  - The lower threshold of the positive activation. This parameter is used to threshold the activation map.
- `offscreen`: boolean, optional
  - If True, Mayavi attempts to plot offscreen. Will work only with VTK >= 5.2.
- `vmin`: float, optional
  - The minimal value, for the colormap
vmax : float, optional
The maximum value, for the colormap
cmap : a callable, or a pylab colormap
A callable returning a (n, 4) array for n values between 0 and 1 for the colors. This can be for instance a pylab colormap.

Notes
If you are using a VTK version below 5.2, there is no way to avoid opening a window during the rendering under Linux. This is necessary to use the graphics card for the rendering. You must maintain this window on top of others and on the screen.

8.3.2 nipy.labs.viz_tools.maps_3d.plot_anat_3d

nipy.labs.viz_tools.maps_3d.plot_anat_3d(anat=None, anat_affine=None, scale=1, sulci_opacity=0.5, gyri_opacity=0.3, opacity=None, skull_percentile=78, wm_percentile=79, outline_color=None)

3D anatomical display

Parameters skull_percentile : float, optional
The percentile of the values in the image that delimit the skull from the outside of the brain. The smaller the fraction of you field of view is occupied by the brain, the larger this value should be.

wm_percentile : float, optional
The percentile of the values in the image that delimit the white matter from the grey matter. Typical this is skull_percentile + 1

For more versatile visualizations the core idea is that given a 3D map and an affine, the data is exposed in Mayavi as a volumetric source, with world space coordinates corresponding to figure coordinates. Visualization modules can be applied on this data source as explained in the Mayavi manual

affine_img_src(data, affine[, scale, name, ...]) Make a Mayavi source defined by a 3D array and an affine, for

8.3.3 nipy.labs.viz_tools.maps_3d.affine_img_src

nipy.labs.viz_tools.maps_3d.affine_img_src(data, affine, scale=1, name='AffineImage', reverse_x=False)

Make a Mayavi source defined by a 3D array and an affine, for which the voxel of the 3D array are mapped by the affine.

Parameters data: 3D ndarray:
The data arrays
affine: (4 x 4) ndarray:
The (4 x 4) affine matrix relating voxels to world coordinates.
scale: float, optional:
An optional addition scaling factor.
name: string, optional :

    The name of the Mayavi source created.

reverse_x: boolean, optional :

    Reverse the x (lateral) axis. Useful to compared with images in radiologic convention.

Notes

The affine should be diagonal.
CHAPTER NINE

GENERATING SIMULATED ACTIVATION MAPS

The module `nipy.labs.utils.simul_multisubject_fmri_dataset` contains various functions to create simulated activation maps in two, three, and four dimensions. A 2D example is `surrogate_2d_dataset()`. The functions can position various activations and add noise, both as background noise and jitter in the activation positions and amplitude.

These functions can be useful to test methods.

9.1 Example

```python
# emacs: -*- mode: python; py-indent-offset: 4; indent-tabs-mode: nil -*-
# vi: set ft=python sts=4 ts=4 sw=4 et:
import numpy as np
import pylab as pl
from nipy.labs.utils.simul_multisubject_fmri_dataset import surrogate_2d_dataset

pos = np.array([[10, 10],
                [14, 20],
                [23, 18]])
ampli = np.array([4, 5, 2])

# First generate some noiseless data
noiseless_data = surrogate_2d_dataset(n_subj=1, noise_level=0, spatial_jitter=0,
                                      signal_jitter=0, pos=pos, ampli=ampli)

pl.figure(figsize=(10, 3))
pl.subplot(1, 4, 1)
pl.imshow(noiseless_data[0])
pl.title('Noise-less data')

# Second, generate some group data, with default noise parameters

group_data = surrogate_2d_dataset(n_subj=3, pos=pos, ampli=ampli)

pl.subplot(1, 4, 2)
pl.imshow(group_data[0])
pl.title('Subject 1')
pl.subplot(1, 4, 3)
```
9.2 Function documentation

\texttt{surrogate\_2d\_dataset(n\_subj=10, shape=(30,30), sk=1.0, noise\_level=1.0, pos=array([[6, 7], [10, 10], [15, 10]]), ampli=array([3, 4, 4]), spatial\_jitter=1.0, signal\_jitter=1.0, width=5.0, width\_jitter=0, out\_text\_file=None, out\_image\_file=None, seed=False)}

Create surrogate (simulated) 2D activation data with spatial noise

\textbf{Parameters} \texttt{n\_subj: integer, optionnal}:

The number of subjects, ie the number of different maps generated.

\texttt{shape=(30,30): tuple of integers, :}

the shape of each image

\texttt{sk: float, optionnal}:

Amount of spatial noise smoothness.

\texttt{noise\_level: float, optionnal}:

Amplitude of the spatial noise. amplitude=noise\_level

\texttt{pos: 2D ndarray of integers, optionnal}:

x, y positions of the various simulated activations.

\texttt{ampli: 1D ndarray of floats, optionnal}:

Respective amplitude of each activation

\texttt{spatial\_jitter: float, optionnal}:

Random spatial jitter added to the position of each activation, in pixel.

\texttt{signal\_jitter: float, optionnal}:

Random amplitude fluctuation for each activation, added to the amplitude specified by \texttt{ampli}

\texttt{width: float or ndarray, optionnal}:

Width of the activations
width_jitter: float :
    Relative width jitter of the blobs

out_text_file: string or None, optional :
    If not None, the resulting array is saved as a text file with the given file name

out_image_file: string or None, optional :
    If not None, the resulting array is saved as a nifti file with the given file name.

seed=False: int, optional :
    If seed is not False, the random number generator is initialized at a certain value

Returns  dataset: 3D ndarray :
    The surrogate activation map, with dimensions (n_subj,) + shape

Create surrogate (simulated) 3D activation data with spatial noise.

Parameters  n_subj: integer, optional :
    The number of subjects, i.e., the number of different maps generated.

shape=(20,20,20): tuple of 3 integers, :
    the shape of each image

mask=None: Nifti1Image instance, :
    referential- and mask-defining image (overrides shape)

sk: float, optional :
    Amount of spatial noise smoothness.

noise_level: float, optional :
    Amplitude of the spatial noise. amplitude=noise_level

pos: 2D ndarray of integers, optional :
    x, y positions of the various simulated activations.

ampli: 1D ndarray of floats, optional :
    Respective amplitude of each activation
spatial_jitter: float, optionnal :
Random spatial jitter added to the position of each activation, in pixel.

signal_jitter: float, optionnal :
Random amplitude fluctuation for each activation, added to the amplitude specified by ampli

width: float or ndarray, optionnal :
Width of the activations

out_text_file: string or None, optionnal :
If not None, the resulting array is saved as a text file with the given file name

out_image_file: string or None, optionnal :
If not None, the resulting is saved as a nifti file with the given file name.

seed=False: int, optionnal :
If seed is not False, the random number generator is initialized at a certain value

Returns dataset: 3D ndarray :
The surrogate activation map, with dimensions (n_subj,) + shape

Create surrogate (simulated) 3D activation data with spatial noise.

Parameters

shape = (20, 20, 20): tuple of integers, :
the shape of each image

mask=None: brifti image instance, :
referential- and mask- defining image (overrides shape)

n_scans: int, optional, :
number of scans to be simlulated overrided by the design matrix

n_sess: int, optional, :
the number of simulated sessions

dmtx: array of shape(n_scans, n_rows), :
the design matrix

sk: float, optionnal :
Amount of spatial noise smoothness.

noise_level: float, optionnal :
Amplitude of the spatial noise. amplitude=noise_level)

**signal_level**: float, optional :
Amplitude of the signal

**out_image_file**: string or list of strings or None, optional :
If not None, the resulting is saved as (set of) nifti file(s) with the given file path(s)

**seed**: False: int, optional :
If seed is not False, the random number generator is initialized at a certain value

**Returns**

**dataset**: a list of n_sess ndarray of shape :

(shape[0], shape[1], shape[2], n_scans) The surrogate activation map
VOLUMETRIC DATA STRUCTURES

Volumetric data structures expose numerical values embedded in a world space. For instance, a volume could expose the T1 intensity, as acquired in scanner space, or the BOLD signal in MNI152 template space. The values can be multi-dimensional, in the case of a BOLD signal, the fMRI signal would correspond to a time series at each position in world space.

10.1 The image structure: VolumeImg

The structure most often used in neuroimaging is the VolumeImg. It corresponds, for instance, to the structure used in the Nifti files. This structure stores data as an n-dimensional array, with n being at least 3, alongside with the necessary information to map it to world space.

**definition** A volume-image (class: VolumeImg) is a volumetric datastructure given by data points lying on a regular grid: this structure is a generalization of an image in 3D. The voxels, vertices of the grid, are mapped to coordinates by an affine transformation. As a result, the grid is regular and evenly-spaced, but may not be orthogonal, and the spacing may differ in the 3 directions.
The data is exposed in a multi dimensional array, with the 3 first axis corresponding to spatial directions. A complete description of this object can be found on the page: VolumeImg.

### 10.2 Useful methods on volume structures

Any general volume structures will implement methods for querying the values and changing world space (see the VolumeField documentation for more details):

- `VolumeField.values_in_world(x, y, z[, ...])` Return the values of the data at the world-space positions given by `x`, `y`, `z`
- `VolumeField.composed_with_transform(...)` Return a new image embedding the same data in a different word space using

#### 10.2.1 nipy.labs.datasets.volumes.volume_field.VolumeField.values_in_world

`VolumeField.values_in_world(x, y, z, interpolation=None)`

Return the values of the data at the world-space positions given by `x`, `y`, `z`

**Parameters**

- `x`: number or ndarray
  - `x` positions in world space, in other words milimeters
- `y`: number or ndarray
  - `y` positions in world space, in other words milimeters. The shape of `y` should match the shape of `x`
- `z`: number or ndarray
z positions in world space, in other words millimeters. The shape of z should match the shape of x

**interpolation**: None, ‘continuous’ or ‘nearest’, optional

Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

**Returns values**: number or ndarray

Data values interpolated at the given world position. This is a number or an ndarray, depending on the shape of the input coordinate.

### 10.2.2 nipy.labs.datasets.volumes.volume_field.VolumeField.composed_with_transform

```python
VolumeField.composed_with_transform(w2w_transform)
```

Return a new image embedding the same data in a different word space using the given world to world transform.

**Parameters**

- **w2w_transform**: transform object

  The transform object giving the mapping between the current world space of the image, and the new word space.

**Returns**

- **remapped_image**: nipy image

  An image containing the same data, expressed in the new world space.

Also, as volumes structure may describe the spatial data in various way, you can easily to convert to a *VolumeImg*, i.e. a regular grid, for instance to do implement an algorithm on the grid such as spatial smoothing:

```python
VolumeField.as_volume_img([affine, shape, ...])
```

Resample the image to be an image with the data points lying on a regular grid with an affine mapping to the word space (a nipy *VolumeImg*).

**Parameters**

- **affine**: 4x4 or 3x3 ndarray, optional

  Affine of the new voxel grid or transform object pointing to the new voxel coordinate grid. If a 3x3 ndarray is given, it is considered to be the rotation part of the affine, and the best possible bounding box is calculated, in this case, the shape argument is not used. If None is given, a default affine is provided by the image.

- **shape**: (n_x, n_y, n_z), tuple of integers, optional

  The shape of the grid used for sampling, if None is given, a default affine is provided by the image.

- **interpolation**: None, ‘continuous’ or ‘nearest’, optional

  Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

**Returns**

- **resampled_image**: nipy *VolumeImg*

  New nipy *VolumeImg* with the data sampled on the grid defined by the affine and shape.
Notes

The coordinate system of the image is not changed: the returned image points to the same world space.

Finally, different structures can embed the data differently in the same world space, for instance with different resolution. You can resample one structure on another using:

\[ \text{VolumeField.resampled_to_img}(\text{target_image}, \ldots) \]  

Resample the volume to be sampled similarly than the target volumetric structure.

10.2.4 nipy.labs.datasets.volumes.volume_field.VolumeField.resampled_to_img

VolumeField.\text{resampled_to_img}(\text{target_image}, \text{interpolation}=\text{None})

Resample the volume to be sampled similarly than the target volumetric structure.

Parameters

- \text{target_image} : nipy volume
  - Nipy volume structure onto the grid of which the data will be resampled.

- \text{interpolation} : None, ‘continuous’ or ‘nearest’, optional
  - Interpolation type used when calculating values in different world spaces. If None, the volume’s interpolation logic is used.

Returns

- \text{resampled_image} : nipy_image
  - New nipy image with the data resampled.

Notes

Both the target image and the original image should be embedded in the same world space.

FIXME: Examples would be good here, but first we need io and template data to be wired with datasets.

10.3 More general data structures

The \text{VolumeImg} is the most commonly found volume structure, and the simplest to understand, however, volumetric data can be described in more generic terms, and for performance reason it might be interesting to use other objects.

Here, we give a list of the nipy volumetric data structures, from most specific, to most general. When you deal with volume structures in your algorithms, depending on which volume structure class you are taking as an input, you can assume different properties of the data. You can always use \text{VolumeImg.as_volume_img()} to cast the volume structure in a \text{VolumeImg} that is simple to understand and easy to work with, but it may not be necessary.

10.3.1 Implemented classes

Implemented classes (or \text{concrete} classes) are structures that you can readily use directly from nipy.

\text{VolumeGrid} In a \text{VolumeGrid}, the data points are sampled on a 3D grid, but unlike for a \text{VolumeImg}, grid may not be regular. For instance, it can be a grid that has been warped by a non-affine transformation. Like with the \text{VolumeImg}, the data is exposed in a multi dimensional array, with the 3 first axis corresponding to spatial directions.
10.3.2 Abstract classes

Abstract classes cannot be used because they are incompletely implemented. They serve as to define the interface: the type of objects that you can use, or how you can extend nipy by exposing the same set of methods and attributes (the interface).

**VolumeData** In this volumetric structure, the data is sampled for some points in the world space. The object knows how to interpolate between these points. The underlying values are stored in a multidimensional array-like object that can be indexed and sliced.
This is an abstract base class: it defines an interface, but is not fully functional, and can be used only via its children class (such as `VolumeGrid` or `VolumeImg`).

**VolumeField** This is the most general volumetric structure (base class): all the nipy volume expose this interface. This structure does not make any assumptions on how the values are internal represented, they may, for instance, be represented as a function, rather than as data points, or as a data structure that is not an array, such as a graph.
This is also an abstract base class: it defines the core nipy volumetric data structure interface: you can rely on all the methods documented for this class in any nipy data structure.
Part III

Developer Guide
DEVELOPMENT QUICKSTART

11.1 Source Code

NIPY uses github for our code hosting. For immediate access to the source code, see the nipy github site.

11.2 Checking out the latest version

To check out the latest version of nipy you need git:

git clone git://github.com/nipy/nipy.git

There are two methods to install a development version of nipy. For both methods, build the extensions in place:

python setup.py build_ext --inplace

Then you can either:

1. Create a symbolic link in your site-packages directory to the inplace build of your source. The advantage of this method is it does not require any modifications of your PYTHONPATH.

2. Place the source directory in your PYTHONPATH.

With either method, all of the modifications made to your source tree will be picked up when nipy is imported.

11.3 Getting data files

See data_files.

11.4 Guidelines

We have adopted many developer guidelines in an effort to make development easy, and the source code readable, consistent and robust. Many of our guidelines are adopted from the scipy / numpy community. We welcome new developers to the effort, if you’re interested in developing code or documentation please join the nipy mailing list and introduce yourself. If you plan to do any code development, we ask that you take a look at the following guidelines. We do our best to follow these guidelines ourselves:

- How to write documentation: Documentation is critical. This document describes the documentation style, syntax, and tools we use.
• **Numpy/Scipy Coding Style Guidelines**: This is the coding style we strive to maintain.
• **Development workflow**: This describes our process for version control.
• **Testing**: We’ve adopted a rigorous testing framework.
• **Optimization**: “premature optimization is the root of all evil.”

### 11.5 Submitting a patch

The preferred method to submit a patch is to create a branch of nipy on your machine, modify the code and make a patch or patches. Then email the nipy mailing list and we will review your code and hopefully apply (merge) your patch. See the instructions for *Making patches*.

If you do not wish to use git and github, please feel free to file a bug report and submit a patch or email the nipy mailing list.

### 11.6 Bug reports

If you find a bug in nipy, please submit a bug report at the nipy bugs github site so that we can fix it.
DEVELOPER INSTALLS FOR DIFFERENT DISTRIBUTIONS

12.1 Debian / Ubuntu developer install

12.1.1 Dependencies

See Download and Install for the installation instructions. Since NiPy is provided within stock distribution (main component of Debian, and universe of Ubuntu), to install all necessary requirements it is enough to:

```bash
sudo apt-get build-dep python-nipy
```

**Note:** Above invocation assumes that you have references to Source repository listed with deb-src prefixes in your apt .list files.

Otherwise, you can revert to manual installation with:

```bash
sudo apt-get build-essential
sudo apt-get install python-dev
sudo apt-get install python-numpy python-numpy-dev python-scipy
sudo apt-get install liblapack-dev
sudo apt-get install python-sympy
```

12.1.2 Useful additions

Some functionality in NiPy requires additional modules:

```bash
sudo apt-get install ipython
sudo apt-get install python-matplotlib
sudo apt-get install mayavi2
```

For getting the code via version control:

```bash
sudo apt-get install git-core
```

Then follow the instructions at Submitting a patch.

And for easier control of multiple Python modules installations (e.g. different versions of IPython):
sudo apt-get install virtualenvwrapper

12.2 Fedora developer install

See Download and Install
This assumes a recent Fedora (>=10) version. It may work for earlier versions - see Download and Install for requirements.
This page may also hold for Fedora-based distributions such as Mandriva and Centos.
Run all the `yum install` commands as root.
Requirements:
- `yum install gcc-c++`
- `yum install python-devel`
- `yum install numpy scipy`
- `yum install sympy`
- `yum install atlas-devel`

Options:
- `yum install ipython`
- `yum install python-matplotlib`

For getting the code via version control:
- `yum install git-core`

Then follow the instructions at Submitting a patch

12.3 Development install on windows

12.3.1 The easy way - a super-package
The easiest way to get the dependencies is to install PythonXY or the Enthought Tool Suite. This gives you MinGW, Python, Numpy, Scipy, ipython and matplotlib (and much more).

12.3.2 The hard way - by components
If instead you want to do it by component, try the instructions below.
Requirements:
- Download and install MinGW
- Download and install the windows binary for Python
- Download and install the Numpy and Scipy binaries
- Download and install Sympy
Options:
- Download and install ipython, being careful to follow the windows installation instructions
• Download and install matplotlib

Alternatively, if you are very brave, you may want to install numpy / scipy from source - see our maybe out of date windows_scipy_build for details.

12.3.3 Getting and installing NIPY

You will next need to get the NIPY code via version control:

• Download and install the windows binary for git

• Go to the windows menu, find the git menu, and run git in a windows terminal.

You should now be able to follow the instructions in Submitting a patch, but with the following modifications:

12.3.4 Running the build / install

Here we assume that you do not have the Microsoft visual C tools, you did not use the ETS package (which sets the compiler for you) and are using a version of MinGW to compile NIPY.

First, for the python setup.py steps, you will need to add the --compiler=mingw32 flag, like this:

```
python setup.py build --compiler=mingw32 install
```

Note that, with this setup you cannot do inplace (developer) installs (like python setup.py build_ext --inplace) because of a six-legged python packaging feature that does not allow the compiler options (here --compiler=mingw32) to be passed from the build_ext command.

If you want to be able to do that, add these lines to your distutils.cfg file

```ini
[build]
compiler=mingw32

[config]
compiler = mingw32
```

See http://docs.python.org/install/#inst-config-files for details on this file. After you’ve done this, you can run the standard python setup.py build_ext --inplace command.

The command line from Windows

The default windows XP command line cmd is very basic. You might consider using the Cygwin bash shell, or you may want to use the ipython shell to work in. For system commands use the ! escape, like this, from the ipython prompt:

```
!python setup.py build --compiler=mingw32
```
CHAPTER
THIRTEEN

DEVELOPMENT GUIDELINES

13.1 How to write documentation

Nipy uses the Sphinx documentation generating tool. Sphinx translates reST formatted documents into html and pdf documents. All our documents and docstrings are in reST format, this allows us to have both human-readable docstrings when viewed in ipython, and web and print quality documentation.

13.1.1 Building the documentation

You need to have Sphinx (version 0.6.2 or above) and graphviz (version 2.20 or greater).

The Makefile (in the top-level doc directory) automates the generation of the documents. To make the HTML documents:

```make html```

For PDF documentation do:

```make pdf```

The built documentation is then placed in a `build/html` or `build/latex` subdirectories.

For more options, type:

```make help```

13.1.2 Viewing the documentation

We also build our website using sphinx. All of the documentation in the docs directory is included on the website. There are a few files that are website only and these are placed in the www directory. The easiest way to view the documentation while editing is to build the website and open the local build in your browser:

```make web```

Then open www/build/html/index.html in your browser.

13.1.3 Syntax

Please have a look at our Sphinx Cheat Sheet for examples on using Sphinx and reST in our documentation.

The Sphinx website also has an excellent sphinx rest primer.
Additional reST references::

• reST primer
• reST quick reference

Consider using emacs for editing rst files - see ReST mode

13.1.4 Style

Nipy has adopted the numpy documentation standards. The numpy coding style guideline is the main reference for how to format the documentation in your code. It’s also useful to look at the source reST file that generates the coding style guideline.

Numpy has a detailed example for writing docstrings.

13.1.5 Documentation Problems

See our Documentation FAQ if you are having problems building or writing the documentation.

13.2 Sphinx Cheat Sheet

Wherein I show by example how to do some things in Sphinx (you can see a literal version of this file below in This file)

13.2.1 Making a list

It is easy to make lists in rest

Bullet points

This is a subsection making bullet points

• point A
• point B
• point C

Enumerated points

This is a subsection making numbered points

1. point A
2. point B
3. point C
13.2.2 Making a table

This shows you how to make a table – if you only want to make a list see Making a list.

<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>John D Hunter</td>
<td>40</td>
</tr>
<tr>
<td>Cast of Thousands</td>
<td>41</td>
</tr>
<tr>
<td>And Still More</td>
<td>42</td>
</tr>
</tbody>
</table>

13.2.3 Making links

Cross-references sections and documents

Use reST labels to cross-reference sections and other documents. The mechanism for referencing another reST document or a subsection in any document, including within a document are identical. Place a reference label above the section heading, like this:

```
.. _sphinx_helpers:
====================
Sphinx Cheat Sheet
====================
```

Note the blank line between the reference label and the section heading is important!

Then refer to the reference label in another document like this:

```
:ref:'sphinx_helpers'
```

The reference is replaced with the section title when Sphinx builds the document while maintaining the linking mechanism. For example, the above reference will appear as Sphinx Cheat Sheet. As the documentation grows there are many references to keep track of.

For documents, please use a reference label that matches the file name. For sections, please try and make the reference label something meaningful and try to keep abbreviations limited. Along these lines, we are using underscores for multiple-word reference labels instead of hyphens.

Sphinx documentation on Cross-referencing arbitrary locations has more details.

External links

For external links you are likely to use only once, simple include the like in the text. This link to google was made like this:

```
'google <http://www.google.com>'
```

For external links you will reference frequently, we have created a links_names.txt file. These links can then be used throughout the documentation. Links in the links_names.txt file are created using the reST reference syntax:

```
.. _targetname: http://www.external_website.org
```

To refer to the reference in a separate reST file, include the links_names.txt file and refer to the link through it's target name. For example, put this include at the bottom of your reST document:

```
.. include:: ../links_names.txt
```
Links to classes, modules and functions

You can also reference classes, modules, functions, etc that are documented using the sphinx autodoc facilites. For example, see the module `matplotlib.backend_bases` documentation, or the class `LocationEvent`, or the method `mpl_connect()`.

13.2.4 ipython sessions

Michael Droettboom contributed a sphinx extension which does pygments syntax highlighting on ipython sessions

```
In [69]: lines = plot([1,2,3])

In [70]: setp(lines)
   ...: alpha: float
   ...: animated: [True | False]
   ...: antialiased or aa: [True | False]
   ...: snip
```

This support is included in this template, but will also be included in a future version of Pygments by default.

13.2.5 Formatting text

You use inline markup to make text *italics*, **bold**, or `monotype`.

You can represent code blocks fairly easily:

```python
import numpy as np
x = np.random.rand(12)
```

Or literally include code:

```
# emacs: -*- mode: python; py-indent-offset: 4; indent-tabs-mode: nil -*-
# vi: set ft=python sts=4 ts=4 sw=4 et:
import matplotlib.pyplot as plt
plt.plot([1,2,3], [4,5,6])
plt.ylabel('some more numbers')
```

13.2.6 Using math

In sphinx you can include inline math \(x \leftarrow y \forall y x - y \) or display math

\[
W^{3\beta}_{\delta_1 \rho_1, \sigma_2} = U^{3\beta}_{\delta_1 \rho_1} + \frac{1}{8\pi^2} \int_{\alpha_2} d\alpha_2 \left[ \frac{U^{2\beta}_{\delta_1 \rho_1, \sigma_2} - \alpha_2^2 U^{1\beta}_{\rho_1, \sigma_2}}{U^{03}_{\rho_1, \sigma_2}} \right]
\]

This documentation framework includes a Sphinx extension, `sphinxext/mathmpl.py`, that uses matplotlib to render math equations when generating HTML, and LaTeX itself when generating a PDF. This can be useful on systems that have matplotlib, but not LaTeX, installed. To use it, add `mathpng` to the list of extensions in `conf.py`.

Current SVN versions of Sphinx now include built-in support for math. There are two flavors:

- `pynmath`: uses dvipng to render the equation
• jsmath: renders the math in the browser using Javascript

To use these extensions instead, add `sphinx.ext.pngmath` or `sphinx.ext.jsmath` to the list of extensions in `conf.py`.

All three of these options for math are designed to behave in the same way.

### 13.2.7 Inserting matplotlib plots

Inserting automatically-generated plots is easy. Simply put the script to generate the plot in any directory you want, and refer to it using the `plot` directive. All paths are considered relative to the top-level of the documentation tree.

To include the source code for the plot in the document, pass the `include-source` parameter:

```python
.. plot:: devel/guidelines/elegant.py
   :include-source:
```

In the HTML version of the document, the plot includes links to the original source code, a high-resolution PNG and a PDF. In the PDF version of the document, the plot is included as a scalable PDF.

```python
import matplotlib.pyplot as plt
plt.plot([1,2,3], [4,5,6])
plt.ylabel('some more numbers')
```

![Plot Diagram](image.png)
13.2.8 Emacs helpers

See ReST mode

13.2.9 Inheritance diagrams

Inheritance diagrams can be inserted directly into the document by providing a list of class or module names to the inheritance-diagram directive.

For example:

```
.. inheritance-diagram:: codecs
```

produces:

![Inheritance Diagram](attachment:inheritance.png)

13.2.10 This file

```
.. _sphinx_helpers:
```

------------------
Sphinx Cheat Sheet
------------------

Wherein I show by example how to do some things in Sphinx (you can see a literal version of this file below in :ref:`sphinx_literal`)
.. _making_a_list:

Making a list
-------------

It is easy to make lists in reST

Bullet points
^^^^^^^^^^^^^^

This is a subsection making bullet points

* point A
* point B
* point C

Enumerated points
^^^^^^^^^^^^^^^^^^

This is a subsection making numbered points

#. point A
#. point B
#. point C

.. _making_a_table:

Making a table
--------------

This shows you how to make a table -- if you only want to make a list see :ref:`making_a_list`.

+----------------+-------+
| Name            | Age   |
+----------------+-------+
| John D Hunter   | 40    |
| Cast of Thousands | 41    |
| And Still More  | 42    |
+----------------+-------+

.. _making_links:

Making links
------------

Cross-references sections and documents
^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

Use reST labels to cross-reference sections and other documents. The mechanism for referencing another reST document or a subsection in any
document, including within a document are identical. Place a
*reference label* above the section heading, like this::

.. _sphinx_helpers:

====================
Sphinx Cheat Sheet
====================

Note the blank line between the *reference label* and the section
heading is important!

Then refer to the *reference label* in another
document like this::

:ref:`sphinx_helpers`

The reference is replaced with the section title when Sphinx builds
the document while maintaining the linking mechanism. For example,
the above reference will appear as :ref:`sphinx_helpers`. As the
documentation grows there are many references to keep track of.

For documents, please use a *reference label* that matches the file
name. For sections, please try and make the *reference label* something
meaningful and try to keep abbreviations limited. Along these lines,
we are using *underscores* for multiple-word *reference labels*
instead of hyphens.

Sphinx documentation on 'Cross-referencing arbitrary locations
<http://sphinx.pocoo.org/markup/inline.html#cross-referencing-arbitrary-locations>'_
has more details.

External links
~~~~~~~~~~~~~~

For external links you are likely to use only once, simple include the
like in the text. This link to 'google <http://www.google.com>'_
was made like this::

'google <http://www.google.com>'_

For external links you will reference frequently, we have created a
``links_names.txt`` file. These links can then be used throughout the
documentation. Links in the ``links_names.txt`` file are created
using the 'reST reference
<http://docutils.sourceforge.net/docs/user/rst/quickref.html#hyperlink-targets>'_
syntax::

.. _targetname: http://www.external_website.org

To refer to the reference in a separate reST file, include the
``links_names.txt`` file and refer to the link through it’s target
name. For example, put this include at the bottom of your reST
document::

.. include:: ../links_names.txt

and refer to the hyperlink target::
Links to classes, modules and functions
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

You can also reference classes, modules, functions, etc that are documented using the sphinx `autodoc`
<http://sphinx.pocoo.org/ext/autodoc.html>`_ facilities. For example, see the module `:mod:`matplotlib.backend_bases` documentation, or the class `:class:`~matplotlib.backend_bases.LocationEvent`, or the method `:meth:`~matplotlib.backend_bases.FigureCanvasBase.mpl_connect`.

.. _ipython_highlighting:

**ipython sessions**
-------------------

Michael Droettboom contributed a sphinx extension which does pygments syntax highlighting on ipython sessions

.. sourcecode:: ipython

    In [69]: lines = plot([1,2,3])
    In [70]: setp(lines)
    alpha: float
       animated: [True | False]
       antialiased or aa: [True | False]
       ...snip

This support is included in this template, but will also be included in a future version of Pygments by default.

.. _formatting_text:

**Formatting text**
-------------------

You use inline markup to make text *italics*, **bold**, or `''monotype'``.

You can represent code blocks fairly easily:

    import numpy as np
    x = np.random.rand(12)

Or literally include code:

.. literalinclude:: elegant.py

.. _using_math:

**Using math**
---------------

In sphinx you can include inline math `:math:``x\leftarrow y\forall x\forall`
This documentation framework includes a Sphinx extension, :file:`sphinxext/mathmpl.py`, that uses matplotlib to render math equations when generating HTML, and LaTeX itself when generating a PDF. This can be useful on systems that have matplotlib, but not LaTeX, installed. To use it, add ``mathpng`` to the list of extensions in :file:`conf.py`.

Current SVN versions of Sphinx now include built-in support for math. There are two flavors:

- pngmath: uses dvipng to render the equation
- jsmath: renders the math in the browser using Javascript

To use these extensions instead, add ``sphinx.ext.pngmath`` or ``sphinx.ext.jsmath`` to the list of extensions in :file:`conf.py`.

All three of these options for math are designed to behave in the same way.

Inserting matplotlib plots
--------------------------

Inserting automatically-generated plots is easy. Simply put the script to generate the plot in any directory you want, and refer to it using the ``plot`` directive. All paths are considered relative to the top-level of the documentation tree. To include the source code for the plot in the document, pass the ``include-source`` parameter:

.. plot:: devel/guidelines/elegant.py
   :include-source:

In the HTML version of the document, the plot includes links to the original source code, a high-resolution PNG and a PDF. In the PDF version of the document, the plot is included as a scalable PDF.

.. plot:: devel/guidelines/elegant.py
   :include-source:

Emacs helpers
-------------

See :ref:`rst_emacs`

Inheritance diagrams
---------------------

Inheritance diagrams can be inserted directly into the document by providing a list of class or module names to the ``inheritance-diagram`` directive.

For example::
13.3 Working with *nipy* source code

Contents:

13.3.1 Introduction

These pages describe a git and github workflow for the nipy project. There are several different workflows here, for different ways of working with *nipy*. 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*.

13.3.2 Install git

Overview

<table>
<thead>
<tr>
<th>Distribution</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debian / Ubuntu</td>
<td><code>sudo apt-get install git-core</code></td>
</tr>
<tr>
<td>Fedora</td>
<td><code>sudo yum install git-core</code></td>
</tr>
<tr>
<td>Windows</td>
<td>Download and install msysGit</td>
</tr>
<tr>
<td>OS X</td>
<td>Use the git-osx-installer</td>
</tr>
</tbody>
</table>

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](http://book.git-scm.com/2_installing_git.html)

13.3.3 Following the latest source

These are the instructions if you just want to follow the latest *nipy* 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:

```bash
git clone git://github.com/nipy/nipy.git
```

You now have a copy of the code tree in the new nipy directory.

Updating the code

From time to time you may want to pull down the latest code. Do this with:

```bash
cd nipy
git pull
```

The tree in nipy will now have the latest changes from the initial repository.

13.3.4 Making a patch

You’ve discovered a bug or something else you want to change in nipy - 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

```bash
# 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/nipy.git
# make a branch for your patching
cd nipy
git branch the-fix-im-thinking-of
git checkout the-fix-im-thinking-of
# hack, hack, hack
# Tell git about any new files you’ve made
git add somewhere/tests/test_my_bug.py
# commit work in progress as you go
git commit -am 'BF - added tests for Funny bug'
# hack hack, hack
git commit -am 'BF - added fix for Funny bug'
```
# make the patch files

git format-patch -M -C master

Then, send the generated patch files to the nipy 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 nipy repository:

   ```
git clone git://github.com/nipy/nipy.git
```

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
```

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 nipy 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 nipy repository on github - Making your own copy (fork) of nipy. Then:

```bash
# checkout and refresh master branch from main repo
git checkout 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/nipy.git
# push up any branches you’ve made and want to keep
push origin the-fix-im-thinking-of
```

Then you can, if you want, follow the Development workflow.

13.3.5 Git for development

Contents:

Making your own copy (fork) of nipy

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 nipy 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 nipy

1. Log into your github account.
2. Go to the nipy github home at nipy 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 nipy.
Set up your fork

First you follow the instructions for *Making your own copy (fork) of nipy.*

**Overview**

```bash
git clone git@github.com:your-user-name/nipy.git
cd nipy
git remote add upstream git://github.com/nipy/nipy.git
```

**In detail**

**Clone your fork**

1. Clone your fork to the local computer with `git clone git@github.com:your-user-name/nipy.git`
2. Investigate. Change directory to your new repo: `cd nipy`. 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 `nipy github` repository, so you can merge in changes from trunk.

**Linking your repository to the upstream repo**

```bash
cd nipy
git remote add upstream git://github.com/nipy/nipy.git
```

`upstream` here is just the arbitrary name we’re using to refer to the main `nipy repository at nipy 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:

```text
upstream git://github.com/nipy/nipy.git (fetch)
upstream git://github.com/nipy/nipy.git (push)
origin git@github.com:your-user-name/nipy.git (fetch)
origin git@github.com:your-user-name/nipy.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:

13.3. Working with `nipy` source code
[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

[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
cb = checkout
st = status -a
stat = status -a
br = branch
wdiff = diff --color-words
```

**Editor**  You may also want to make sure that your editor of choice is used

git config --global core.editor vim

**Merging**  To enforce summaries when doing merges (~/.gitconfig file again):

```
[merge]
log = true
```

Or from the command line:

git config --global merge.log true

**Development workflow**

You already have your own forked copy of the nipy repository, by following *Making your own copy (fork) of nipy, 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 nipy development repo. Your master branch will follow the main nipy 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 nipy. 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:

```bash
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:

```bash
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**

```bash
# 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:

```bash
# 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/nipy.
2. Click on the Branch list button:
3. Click on the Compare button for your feature branch - here my-new-feature:

4. If asked, select the base and comparison branch names you want to compare. Usually these will be master and my-new-feature (where that is your feature branch name).

5. At this point you should get a nice summary of the changes. Copy the URL for this, and post it to the nipy mailing list, asking for review. The URL will look something like: http://github.com/your-user-name/nipy/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) and see dot2-dot3.

Asking for your changes to be merged with the main repo

When you are ready to ask for the merge of your code:

1. Go to the URL of your forked repo, say http://github.com/your-user-name/nipy.git.

2. Click on the ‘Pull request’ button:

Enter a message; we suggest you select only nipy as the recipient. The message will go to the nipy mailing list. Please feel free to add others from the list as you like.

Merging from trunk

This updates your code from the upstream nipy github repo.

Overview
# go to your master branch
git checkout master
# pull changes from github
git fetch upstream
# merge from upstream
git merge upstream/master

**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 nipy into your account, as from *Making your own copy (fork) of nipy.*

Then, go to your forked repository github page, say http://github.com/your-user-name/nipy

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/nipy.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](https://github.com) for your github repo.

### 13.3.6 git resources

#### Tutorials and summaries

- [github help](https://help.github.com) has an excellent series of how-to guides.
- [learn.github](https://learn.github.com) has an excellent series of tutorials
- The [pro git book](https://git-scm.com/book) is a good in-depth book on git.
- A [git cheat sheet](https://git-scm.com/docs) is a page giving summaries of common commands.
- The [git user manual](https://git-scm.com/docs)
- The [git tutorial](https://git-scm.com/book)
- The [git community book](https://github.com/choosegitter/choosegitter)
- [git ready](https://git-scm.com/doc) - a nice series of tutorials
- [git casts](https://git-casts.com) - video snippets giving git how-tos.
- [git magic](https://github.com/gitguides/git-magic) - extended introduction with intermediate detail
- The [git parable](https://git-scm.com/book) is an easy read explaining the concepts behind git.
- Fernando Perez’ [git page](https://learnforever.house/) - many links and tips
- A good but technical page on [git concepts](https://git-scm.com/book)
- [git svn crash course](https://git-scm.com/doc): 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`

13.4 Commit message codes

Please prefix all commit summaries with one (or more) of the following labels. This should help others to easily classify the commits into meaningful categories:

- **BF**: bug fix
- **RF**: refactoring
- **ENH**: new feature or extended functionality
- **BW**: addresses backward-compatibility
- **OPT**: optimization
- **BK**: breaks something and/or tests fail
- **DOC**: for all kinds of documentation related commits
- **TEST**: for adding or changing tests
- **STY**: PEP8 conformance, whitespace changes etc that do not affect function.
- **WIP**: Work in progress; please try and avoid using this one, and rebase incomplete changes into functional units using e.g. `git rebase -i`
So your commit message might look something like this:

TEST: relax test threshold slightly

Attempted fix for failure on windows test run when arrays are in fact very close (within 6 dp).

Keeping up a habit of doing this is useful because it makes it much easier to see at a glance which changes are likely to be important when you are looking for sources of bugs, fixes, large refactorings or new features.

13.5 Pull request codes

When you submit a pull request to github, github will ask you for a summary. If your code is not ready to merge, but you want to get feedback, please consider using WIP - me working on image design or similar for the title of your pull request. That way we will all know that it’s not yet ready to merge and that you may be interested in more fundamental comments about design.

When you think the pull request is ready to merge, change the title (using the Edit button) to something like MRG - my work on image design.

13.6 Testing

Nipy uses the Numpy test framework which is based on nose. If you plan to do much development you should familiarize yourself with nose and read through the numpy testing guidelines.

13.6.1 Writing tests

Test files

The numpy testing framework and nipy extensions are imported with one line in your test module:

```
from nipy.testing import *
```

This imports all the assert_* functions you need like assert_equal, assert_raises, assert_array_almost_equal etc..., numpy's rand function, and the numpy test decorators: knownfailure, slow, skipif, etc...

Please name your test file with the test_ prefix followed by the module name it tests. This makes it obvious for other developers which modules are tested, where to add tests, etc... An example test file and module pairing:

```
nipy/core/reference/coordinate_system.py
nipy/core/reference/tests/test_coordinate_system.py
```

All tests go in a test subdirectory for each package.

Temporary files

If you need to create a temporary file during your testing, you could use one of these three methods, in order of convenience:
1. **StringIO**

   StringIO creates an in memory file-like object. The memory buffer is freed when the file is closed. This is the preferred method for temporary files in tests.

2. **nibabel.tmpdirs.InTemporaryDirectory context manager.**

   This is a convenient way of putting you into a temporary directory so you can save anything you like into the current directory, and feel fine about it after. Like this:

   ```python
   from ..tmpdirs import InTemporaryDirectory
   with InTemporaryDirectory():
       f = open('myfile', 'wt')
       f.write('Anything at all')
       f.close()
   
   One thing to be careful of is that you may need to delete objects holding onto the file before you exit the with statement, otherwise Windows may refuse to delete the file.

3. **tempfile.mkstemp**

   This will create a temporary file which can be used during testing. There are parameters for specifying the filename, prefix, and suffix.

   **Note:** The tempfile module includes a convenience function NamedTemporaryFile which deletes the file automatically when it is closed. However, whether the files can be opened a second time varies across platforms and there are problems using this function on Windows.

   **Example:**

   ```python
   from tempfile import mkstemp
   try:
       fd, name = mkstemp(suffix='.nii.gz')
       tmpfile = open(name)
       save_image(fake_image, tmpfile.name)
       tmpfile.close()
   finally:
       os.unlink(name)  # This deletes the temp file
   
   Please don’t just create a file in the test directory and then remove it with a call to os.remove. For various reasons, sometimes os.remove doesn’t get called and temp files get left around.

**Many tests in one test function**

To keep tests organized, it’s best to have one test function correspond to one class method or module-level function. Often though, you need many individual tests to thoroughly cover (100% coverage) the method/function. This calls for a generator function. Use a yield statement to run each individual test, independent from the other tests. This prevents the case where the first test fails and as a result the following tests don’t get run.

This test function executes four independent tests:

```python
def test_index():
    cs = CoordinateSystem('ijk')
    yield assert_equal, cs.index('i'), 0
    yield assert_equal, cs.index('j'), 1
    yield assert_equal, cs.index('k'), 2
    yield assert_raises, ValueError, cs.index, 'x'
```
Suppress **warnings** on test output

In order to reduce noise when running the tests, consider suppressing *warnings* in your test modules. This can be done in the module-level setup and teardown functions:

```python
import warnings
...

def setup():
    # Suppress warnings during tests to reduce noise
    warnings.simplefilter("ignore")

def teardown():
    # Clear list of warning filters
    warnings.resetwarnings()
```

### 13.6.2 Running tests

**Running the full test suite**

For our tests, we have collected a set of fmri imaging data which are required for the tests to run. To do this, download the latest example data and template package files from NIPY data packages. See data-files.

**Running individual tests**

You can also run nose from the command line with a variety of options. To test an individual module:

```
nosetests test_image.py
```

To test an individual function:

```
nosetests test_module:test_function
```

To test a class:

```
nosetests test_module:TestClass
```

To test a class method:

```
nosetests test_module:TestClass.test_method
```

Verbose mode (-v option) will print out the function names as they are executed. Standard output is normally suppressed by nose, to see any print statements you must include the -s option. In order to get a “full verbose” output, call nose like this:

```
nosetests -sv test_module.py
```

To include doctests in the nose test:

```
nosetests -sv --with-doctest test_module.py
```

For details on all the command line options:

```
nosetests --help
```
13.6.3 Coverage Testing

Coverage testing is a technique used to see how much of the code is exercised by the unit tests. It is important to remember that a high level of coverage is a necessary but not sufficient condition for having effective tests. Coverage testing can be useful for identifying whole functions or classes which are not tested, or for finding certain conditions which are never tested.

This is an excellent task for nose - the automated test runner we are using. Nose can run the python coverage tester. First make sure you have the coverage tester installed on your system. Download the tarball from the link, extract and install python setup.py install. Or on Ubuntu you can install from apt-get:

```bash
sudo apt-get install python-coverage
```

Run nose with coverage testing arguments:

```bash
nosetests -sv --with-coverage path_to_code
```

For example, this command:

```bash
nosetests -sv --with-coverage test_coordinate_map.py
```

will report the following:

<table>
<thead>
<tr>
<th>Name</th>
<th>Stmts</th>
<th>Exec</th>
<th>Cover</th>
<th>Missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>nipy</td>
<td>21</td>
<td>14</td>
<td>66%</td>
<td>70-74, 88-89</td>
</tr>
<tr>
<td>nipy.core</td>
<td>4</td>
<td>4</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>nipy.core.reference</td>
<td>8</td>
<td>8</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>nipy.core.reference.array_coors</td>
<td>100</td>
<td>90</td>
<td>90%</td>
<td>133-134, 148-151, 220, 222, 235, 242</td>
</tr>
<tr>
<td>nipy.core.reference.coordinate_map</td>
<td>188</td>
<td>187</td>
<td>99%</td>
<td>738</td>
</tr>
<tr>
<td>nipy.core.reference.coordinate_system</td>
<td>61</td>
<td>61</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>nipy.core.reference.slices</td>
<td>34</td>
<td>34</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>nipy.core.transforms</td>
<td>0</td>
<td>0</td>
<td>100%</td>
<td></td>
</tr>
<tr>
<td>nipy.core.transforms.affines</td>
<td>14</td>
<td>14</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

The coverage report will cover any python source module imported after the start of the test. This can be noisy and difficult to focus on the specific module for which you are writing nosetests. For instance, the above report also included coverage of most of numpy. To focus the coverage report, you can provide nose with the specific package you would like output from using the --cover-package. For example, in writing tests for the coordinate_map module:

```bash
nosetests --with-coverage --cover-package=nipy.core.reference.coordinate_map test_coordinate_map.py
```

Since that’s a lot to type, I wrote a tool called sneeze to that simplifies coverage testing with nose.

**Sneeze**

Sneeze runs nose with coverage testing and reports only the package the test module is testing. It requires the test module follow a simple naming convention:

1. Prefix test_
2. The package name you are testing
3. Suffix .py

For example, the test module for the coordinate_map module is named test_coordinate_map.py. Then testing coverage is as simple as:

```bash
sneeze.py test_coordinate_map.py
```
Sneeze is included in the tools directory in the nipy source. Simply run the setup.py to install sneeze in your local bin directory.

### 13.7 Debugging

Some options are:

#### 13.7.1 Run in ipython

As in:

```
In [1]: run mymodule.py  
... (somecrash)  
In [2]: %debug
```

Then diagnose, using the workspace that comes up, which has the context of the crash.

You can also do:

```
In [1] %pdb on  
In [2]: run mymodule.py  
... (somecrash)
```

At that point you will be automatically dropped into the the workspace in the context of the error. This is very similar to the matlab dbstop if error command.

See the ipython manual, and debugging in ipython for more detail.

#### 13.7.2 Embed ipython in crashing code

Often it is not possible to run the code directly from ipython using the run command. For example, the code may be called from some other system such as sphinx. In that case you can embed. At the point that you want ipython to open with the context available for introspection, add:

```
from IPython.Shell import IPShellEmbed  
ipshell = IPShellEmbed()  
ipshell()  
```

See embedding ipython for more detail.

### 13.8 Optimization

In the early stages of NIPY development, we are focusing on functionality and usability. In regards to optimization, we benefit significantly from the optimized routines in scipy and numpy. As NIPY progresses it is likely we will spend more energy on optimizing critical functions. In our py4science group at UC Berkeley we’ve had several meetings on the various optimization options including ctypes, weave and blitz, and cython. It’s clear there are many good options, including standard C-extensions. However, optimized code tends to be less readable and more difficult to debug and maintain. When we do optimize our code we will first profile the code to determine the offending sections, then optimize those sections. Until that need arises, we will follow the great advice from these fellow programmers:

**Kent Beck:** “First make it work. Then make it right. Then make it fast.”

Donald Knuth on optimization:
“We should forget about small efficiencies, say about 97% of the time: premature optimization is the root of all evil.”

Tim Hochberg, from the Numpy list:

0. Think about your algorithm.
1. Vectorize your inner loop.
2. Eliminate temporaries
3. Ask for help
4. Recode in C.
5. Accept that your code will never be fast.

Step zero should probably be repeated after every other step ;)
• Review and update the release notes. Review and update the Changelog file. Get a partial list of contributors with something like:

```
git log 0.2.0.. | grep '^Author' | cut -d' ' -f 2- | sort | uniq
```

where 0.2.0 was the last release tag name.
Then manually go over `git shortlog 0.2.0..` to make sure the release notes are as complete as possible and that every contributor was recognized.

• Use the opportunity to update the .mailmap file if there are any duplicate authors listed from `git shortlog`.

• Check the examples in python 2 and python 3, by running something like:

```
cd ..
./nipy/tools/run_log_examples.py nipy/examples --log-path=~/tmp/eg_logs
```

in a python 2 and python 3 virtualenv. Review the output in (e.g.) ~/tmp/eg_logs. The output file summary.txt will have the pass file printout that the run_log_examples.py script puts onto stdout while running.

• Check the long_description in nipy/info.py. Check it matches the README in the root directory, maybe with `vim diffthis` command.

• Do a final check on the nipy buildbot

• If you have travis-ci building set up you might want to push the code in its current state to a branch that will build, e.g:

```
git branch -D pre-release-test # in case branch already exists
git co -b pre-release-test
git push origin pre-release-test
```

• Make sure all the .c generated files are up to date with Cython sources with:

```
./tools/nicythize
```

13.10.3 Release checking - buildbots

• Check all the buildbots pass

• Run the builder and review the possibly green output from http://nipy.bic.berkeley.edu/builders/nipy-release-checks

This runs all of:

```
make distclean
python -m compileall .
make sdist-tests
make bdist-egg-tests
make check-version-info
make check-files
```

• You need to review the outputs for errors; at the moment this buildbot builder does not check whether these tests passed or failed.

• `make bdist-egg-tests` may well fail because of a problem with the script tests; if you have a recent (>= Jan 15 2013) nibabel nisext package, you could try instead doing:
Eventually we should update the bdist-egg-tests makefile target.

- make check-version-info checks how the commit hash is stored in the installed files. You should see something like this:

```bash
({'sys_version': '2.6.6 (r266:84374, Aug 31 2010, 11:00:51) 
[GCC 4.0.1 (Apple Inc. build 5493)]
/var/folders/jg/jgfZ12ZXHwGSFKD85xLpLk+++TI/-Tmp-/tmpGPiD3E/pylib/nipy/__init__.pyc
('sys_version': '2.6.6 (r266:84374, Aug 31 2010, 11:00:51) 
[GCC 4.0.1 (Apple Inc. build 5493)]
/Users/mb312/dev_trees/nipy/nipy/__init__.pyc
('sys_version': '2.6.6 (r266:84374, Aug 31 2010, 11:00:51) 
[GCC 4.0.1 (Apple Inc. build 5493)]
```

- make check-files checks if the source distribution is picking up all the library and script files. Look for output at the end about missed files, such as:

```
Missed script files: /Users/mb312/dev_trees/nipy/bin/nib-dicomfs, /Users/mb312/dev_trees/nipy/bin/nifti1_diagnose.py
```

Fix setup.py to carry across any files that should be in the distribution.

- Check the documentation doctests pass from http://nipy.bic.berkeley.edu/builders/nipy-doc-builder

- You may have virtualenvs for different python versions. Check the tests pass for different configurations. If you have pytox and a network connection, and lots of pythons installed, you might be able to do:

```
tox
```

and get tests for python 2.5, 2.6, 2.7, 3.2. I (MB) have my own set of virtualenvs installed and I’ve set them up to run with:

```
tox -e python25,python26,python27,python32,np-1.2.1
```

The trick was only to define these testenv sections in tox.ini.

These two above run with:

```
make tox-fresh
make tox-stale
```

respectively.

The long-hand not-tox way looks like this:

```
workon python26
make sdist-tests
deactivate
```

etc for the different virtualenvs.

### 13.10.4 Doing the release

- The release should now be ready.

- Edit nipy/info.py to set _version_extra to "":commit. Then:

```
make source-release
```

- Once everything looks good, you are ready to upload the source release to PyPi. See setuptools intro. Make sure you have a file \$HOME/.pypirc, of form:
Once everything looks good, upload the source release to PyPi. See setuptools intro:

```bash
python setup.py register
python setup.py sdist --formats=gztar,zip upload
```


- Download the mpkg builds, maybe with:
  ```bash
  scp -r buildbot@nipy.bic.berkeley.edu:nibotmi/public_html/nipy-dist/*.*.mpkg
  ```

Make sure you have `github bdist_mpkg` installed, for the root user. For each mpkg directory, run:

```bash
sudo reown_mpkg nipy-0.3.0.dev-py2.6-macosx10.6.mpkg root admin
zip -r nipy-0.3.0.dev-py2.6-macosx10.6.mpkg.zip nipy-0.3.0.dev-py2.6-macosx10.6.mpkg
```

Upload the mpkg.zip files. (At the moment, these don’t seem to store the scripts - needs more work)

- Tag the release with tag of form 0.3.0:
  ```bash
  git tag -am 'Second main release' 0.3.0
  ```

- Now the version number is OK, push the docs to sourceforge with:
  ```bash
  cd doc
  make upload-stable-web-mysfusername
  ```

  where `mysfusername` is obviously your own sourceforge username.

- Set up maintenance / development branches

  If this is this is a full release you need to set up two branches, one for further substantial development (often called ‘trunk’) and another for maintenance releases.

  - Branch to maintenance:
    ```bash
    git co -b maint/0.2.x
    ```

    Set `_version_extra` back to `.dev` and bump `_version_micro` by 1. Thus the maintenance series will have version numbers like - say - ‘0.2.1.dev’ until the next maintenance release - say ‘0.2.1’. Commit. Don’t forget to push upstream with something like:
    ```bash
    git push upstream maint/0.2.x --set-upstream
    ```
– Start next development series:

```bash
    git co main-master
```

then restore `.dev` to `_version_extra`, and bump `_version_minor` by 1. Thus the development series (‘trunk’) will have a version number here of ‘0.3.0.dev’ and the next full release will be ‘0.3.0’.

If this is just a maintenance release from `maint/0.2.x` or similar, just tag and set the version number to - say - `0.2.1.dev`.

- Push tags:

  ```bash
    git push --tags
  ```

- Announce to the mailing lists.

## 13.11 The ChangeLog

**NOTE:** We have not kepted up with our ChangeLog. This is here for future reference. We will be more diligent with this when we have regular software releases.

If you are a developer with commit access, please fill a proper ChangeLog entry per significant change. The SVN commit messages may be shorter (though a brief summary is appreciated), but a detailed ChangeLog is critical. It gives us a history of what has happened, allows us to write release notes at each new release, and is often the only way to backtrack on the rationale for a change (as the diff will only show the change, not why it happened).

Please skim the existing ChangeLog for an idea of the proper level of detail (you don’t have to write a novel about a patch).

The existing ChangeLog is generated using (X)Emacs’ fantastic ChangeLog mode: all you have to do is position the cursor in the function/method where the change was made, and hit ‘C-x 4 a’. XEmacs automatically opens the ChangeLog file, mark a dated/named point, and creates an entry pre-titled with the file and function name. It doesn’t get any better than this. If you are not using (X)Emacs, please try to follow the same convention so we have a readable, organized ChangeLog.

To get your name in the ChangeLog, set this in your `.emacs` file:

```
  (setq user-full-name "Your Name")  (setq user-mail-address "youraddress@domain.com")
```

Feel free to obfuscate or omit the address, but at least leave your name in. For user contributions, try to give credit by name on patches or significant ideas, but please do an @ -AT- replacement in the email addresses (users have asked for this in the past).
14.1 Nipy roadmap

We plan to release a prototype of NIPY by the Summer of 2009. This will include a full FMRI analysis, 2D visualization, and integration with other packages for spatial processing (SPM and FSL). We will continue to improve our documentation and tutorials with the aim of providing a full introduction to neuroimaging analysis.

We will also extend our collaborations with other neuroimaging groups, integrating more functionality into NIPY and providing better interoperability with other packages. This will include the design and implementation of a pipeline/batching system, integration of registration algorithms, and improved 2D and 3D visualization.

14.2 TODO for nipy development

This document will serve to organize current development work on nipy. It will include current sprint items, future feature ideas, and design discussions, etc...

14.2.1 Documentation

- Create NIPY sidebar with links to all project related websites.
- Create a Best Practices document.
- Create a rst doc for Request a review process.

Tutorials

Tutorials are an excellent way to document and test the software. Some ideas for tutorials to write in our Sphinx documentation (in no specific order):

- Slice timing
- Image resampling
- Image IO
- Registration using SPM/FSL
- FMRI analysis
- Making one 4D image from many 3D images, and vice versa. Document ImageList and FmriImageList.
- Apply SPM registration .mat to a NIPY image.
• Create working example out of this TRAC pca page. Should also be a rest document.
• Add analysis pipeline(s) blueprint.

14.2.2 Bugs

These should be moved to the nipy bug section on github. Placed here until they can be input.

• Fix possible precision error in fixes.scipy.ndimage.test_registration function test_autoalign_nmi_value_2. See FIXME.
• Fix error in test_segment test_texture2 functions (fixes.scipy.ndimage). See FIXME.
• import nipy.algorithms is very slow! Find and fix. The shared library is slow.
• base class for all new-style classes should be object; preliminary search with grin "class +[a-zA-Z0-9]+ *=:

14.2.3 Refactorings

• image.save function should accept filename or file-like object. If I have an open file I would like to be able to pass that in also, instead of fp.name. Happens in test code a lot.
• image._open function should accept Image objects in addition to ndarrays and filenames. Currently the save function has to call np.asarray(img) to get the data array out of the image and pass them to _open in order to create the output image.
• Add dtype options when saving. When saving images it uses the native dtype for the system. Should be able to specify this. in the test_file_roundtrip, self.img is a uint8, but is saved to tmpfile as float64. Adding this would allow us to save images without the scaling being applied.
• In image._open(url, ...), should we test if the “url” is a PyNiftiIO object already? This was in the tests from ‘old code’ and passed:

\[
new = Image(self.img._data, self.img.grid)
\]

img._data is a PyNiftiIO object. It works, but we should verify it’s harmless otherwise prevent it from happening.
• Look at image.merge_image function. Is it still needed? Does it fit into the current api?
• FmriImageList.emptycopy() - Is there a better way to do this? Matthew proposed possibly implementing Gael’s dress/undress metadata example.
• Verify documentation of the image generators. Create a simple example using them.
• Use python 2.5 feature of being able to reset the generator?
• Add test data where volumes contain intensity ramps. Slice with generator and test ramp values.
• Implement fmrimagelist blueprint.

14.2.4 Code Design Thoughts

A central location to dump thoughts that could be shared by the developers and tracked easily.
14.2.5 Future Features

Put ideas here for features nipy should have but are not part of our current development. These features will eventually be added to a weekly sprint log.

- Auto backup script for nipy repos to run as weekly cron job. We should setup a machine to perform regular branch builds and tests. This would also provide an on-site backup.
- See if we can add bz2 support to nifticlib.
- Should image.load have an optional squeeze keyword to squeeze a 4D image with one frame into a 3D image?
These are some developer discussions about design of code in NIPY.

15.1 Understanding voxel and real world mappings

15.1.1 Voxel coordinates and real-world coordinates

A point can be represented by coordinates relative to specified axes. Coordinates are (almost always) numbers - see coordinate systems.

For example, a map grid reference gives a coordinate (a pair of numbers) to a point on the map. The numbers give the respective positions on the horizontal ($x$) and vertical ($y$) axes of the map.

A coordinate system is defined by a set of axes. In the example above, the axes are the $x$ and $y$ axes. Axes for coordinates are usually orthogonal - for example, moving one unit up on the $x$ axis on the map causes no change in the $y$ coordinate - because the axes are at 90 degrees.

In this discussion we’ll concentrate on the three dimensional case. Having three dimensions means that we have a three axis coordinate system, and coordinates have three values. The meaning of the values depend on what the axes are.

Voxel coordinates

Array indexing is one example of using a coordinate system. Let’s say we have a three dimensional array:

$$A = \text{np.arange(24).reshape((2,3,4))}$$

The value 0 is at array coordinate 0, 0, 0:

```python
assert A[0,0,0] == 0
```

and the value 23 is at array coordinate 1, 2, 3:

```python
assert A[1,2,3] == 23
```

(respecting python’s zero-based indexing). If we now say that our array is a 3D volume element array - an array of voxels, then the array coordinate is also a voxel coordinate.

If we want to use numpy to index our array, then we need integer voxel coordinates, but if we use a resampling scheme, we can also imagine non-integer voxel coordinates for $A$, such as $(0.6, 1.2, 1.9)$, and we could use resampling to estimate the value at such a coordinate, given the actual data in the surrounding (integer) points.)
Array / voxel coordinates refer to the array axes. Without any further information, they do not tell us about where the point is in the real world - the world we can measure with a ruler. We refer to array / voxel coordinates with indices \( i, j, k \), where \( i \) is the first value in the 3 value coordinate tuple. For example, if array / voxel point \((1, 2, 3)\) has \( i=1, j=2, k=3 \). We’ll be careful only to use \( i, j, k \) rather than \( x, y, z \), because we are going to use \( x, y, z \) to refer to real-world coordinates.

Real-world coordinates

Real-world coordinates are coordinates where the values refer to real-world axes. A real-world axis is an axis that refers to some real physical space, like low to high position in an MRI scanner, or the position in terms of the subject’s head.

Here we’ll use the usual neuroimaging convention, and that is to label our axes relative to the subject’s head:

- \( x \) has negative values for left and positive values for right
- \( y \) has negative values for posterior (back of head) and positive values for anterior (front of head)
- \( z \) has negative values for the inferior (towards the neck) and positive values for superior (towards the highest point of the head, when standing)

15.1.2 Image index ordering

Background

In general, images - and in particular NIfTI format images, are ordered in memory with the X dimension changing fastest, and the Z dimension changing slowest.

Numpy has two different ways of indexing arrays in memory, C and fortran. With C index ordering, the first index into an array indexes the slowest changing dimension, and the last indexes the fastest changing dimension. With fortran ordering, the first index refers to the fastest changing dimension - X in the case of the image mentioned above.

C is the default index ordering for arrays in Numpy.

For example, let’s imagine that we have a binary block of 3D image data, in standard NIfTI / Analyze format, with the X dimension changing fastest, called \( \text{my.img} \), containing Float32 data. Then we memory map it:

```python
img_arr = memmap(‘my.img’, dtype=float32)
```

When we index this new array, the first index indexes the Z dimension, and the third indexes X. For example, if I want a voxel \( X=3, Y=10, Z=20 \) (zero-based), I have to get this from the array with:

```python
img_arr[20, 10, 3]
```

The problem

Most potential users of NiPy are likely to have experience of using image arrays in Matlab and SPM. Matlab uses Fortran index ordering. For fortran, the first index is the fastest changing, and the last is the slowest-changing. For example, here is how to get voxel \( X=3, Y=10, Z=20 \) (zero-based) using SPM in Matlab:

```matlab
img_arr = spm_read_vols(spm_vol(‘my.img’));
img_arr(4, 11, 21) % matlab indexing is one-based
```

This ordering fits better with the way that we talk about coordinates in functional imaging, as we invariably use XYZ ordered coordinates in papers. It is possible to do the same in numpy, by specifying that the image should have fortran index ordering:
The proposal

Change the default ordering of image arrays to fortran, in order to allow XYZ index ordering. So, change the access to the image array in the image class so that, to get the voxel at X=3, Y=10, Z=20 (zero-based):

```python
img = Image('my.img')
img[3, 10, 20]
```

instead of the current situation, which requires:

```python
img = Image('my.img')
img[20, 10, 3]
```

Summary of discussion

For:

- Fortran index ordering is more intuitive for functional imaging because of conventional XYZ ordering of spatial coordinates, and Fortran index ordering in packages such as Matlab
- Indexing into a raw array is fast, and common in lower-level applications, so it would be useful to implement the more intuitive XYZ ordering at this level rather than via interpolators (see below)
- Standardizing to one index ordering (XYZ) would mean users would not have to think about the arrangement of the image in memory

Against:

- C index ordering is more familiar to C users
- C index ordering is the default in numpy
- XYZ ordering can be implemented by wrapping by an interpolator

Potential problems

Performance penalties

KY commented:

This seems like a good idea to me but I have no knowledge of numpy internals (and even less than none after the numeric/numarray integration). Does anyone know if this will (or definitely will not) incur any kind of obvious performance penalties re. array operations (sans arcane problems like stride issues in huge arrays)?

MB replied:

Note that, we are not proposing to change the memory layout of the image, which is fixed by the image format in e.g NIfTI, but only to index it XYZ instead of ZYX. As far as I am aware, there are no significant performance differences between:

```python
img_arr = memmap('my.img', dtype=float32, order='F')
img_arr[3, 10, 20]
```

```python
img_arr = memmap('my.img', dtype=float32, order='C')
img_arr[5, 4, 3]
```
and:

```python
img_arr = memmap('my.img', dtype=float32, order='F')
img_arr[3,4,5]
```

Happy to be corrected though.

**Clash between default ordering of numpy arrays and nipy images**

C index ordering is the default in numpy, and using fortran ordering for images might be confusing in some circumstances. Consider for example:

```python
img_obj = Image('my.img') # Where the Image class has been changed to implement Fortran ordering
first_z_slice = img_obj[...,0] # returns a Z slice
```

```python
img_arr = memmap('my.img', dtype=float32) # C ordering, the numpy default
img_obj = Image(img_arr)
first_z_slice = img_obj[...,0] # in fact returns an X slice
```

I suppose that we could check that arrays are fortran index ordered in the Image __init__ routine.

**An alternative proposal - XYZ ordering of output coordinates**

JT: Another thought, that is a compromise between the XYZ coordinates and Fortran ordering.

To me, having worked mostly with C-type arrays, when I index an array I think in C terms. But, the Image objects have the “warp” attached to them, which describes the output coordinates. We could insist that the output coordinates are XYZ(T) (or make this an option). So, for instance, if the 4x4 transform was the identity, the following two calls would give something like:

```python
interp = interpolator(img)
img[3,4,5] == interp(5,4,3)
```

This way, users would be sure in the interpolator of the order of the coordinates, but users who want access to the array would know that they would be using the array order on disk...

I see that a lot of users will want to think of the first coordinate as “x”, but depending on the sampling the [0] slice of `img` may be the leftmost or the rightmost. To find out which is which, users will have to look at the 4x4 transform (or equivalently the start and the step). So just knowing the first array coordinate is the “x” coordinate still misses some information, all of which is contained in the transform.

MB replied:

I agree that the output coordinates are very important - and I think we all agree that this should be XYZ(T)?

For the raw array indices - it is very common for people to want to do things to the raw image array - the quickstart examples containing a few - and you usually don’t care about which end of X is left in that situation, only which spatial etc dimension the index refers to.

**15.2 Image index ordering**

**15.2.1 Background**

In general, images - and in particular NIfTI format images, are ordered in memory with the X dimension changing fastest, and the Z dimension changing slowest.
Numpy has two different ways of indexing arrays in memory, C and fortran. With C index ordering, the first index into an array indexes the slowest changing dimension, and the last indexes the fastest changing dimension. With fortran ordering, the first index refers to the fastest changing dimension - X in the case of the image mentioned above.

C is the default index ordering for arrays in Numpy.

For example, let's imagine that we have a binary block of 3D image data, in standard NIfTI / Analyze format, with the X dimension changing fastest, called `my.img`, containing Float32 data. Then we memory map it:

```python
img_arr = memmap('my.img', dtype=float32)
```

When we index this new array, the first index indexes the Z dimension, and the third indexes X. For example, if I want a voxel X=3, Y=10, Z=20 (zero-based), I have to get this from the array with:

```python
img_arr[20, 10, 3]
```

### 15.2.2 The problem

Most potential users of NiPy are likely to have experience of using image arrays in Matlab and SPM. Matlab uses Fortran index ordering. For fortran, the first index is the fastest changing, and the last is the slowest-changing. For example, here is how to get voxel X=3, Y=10, Z=20 (zero-based) using SPM in Matlab:

```python
img_arr = spm_read_vols(spm_vol('my.img'));
img_arr(4, 11, 21)  % matlab indexing is one-based
```

This ordering fits better with the way that we talk about coordinates in functional imaging, as we invariably use XYZ ordered coordinates in papers. It is possible to do the same in numpy, by specifying that the image should have fortran index ordering:

```python
img_arr = memmap('my.img', dtype=float32, order='F')
img_arr[3, 10, 20]
```

### 15.2.3 Native fortran or C indexing for images

We could change the default ordering of image arrays to fortran, in order to allow XYZ index ordering. So, change the access to the image array in the image class so that, to get the voxel at X=3, Y=10, Z=20 (zero-based):

```python
img = load_image('my.img')
img[3, 10, 20]
```

instead of the current situation, which requires:

```python
img = load_image('my.img')
img[20, 10, 3]
```

### For and against fortran ordering

For:

- Fortran index ordering is more intuitive for functional imaging because of conventional XYZ ordering of spatial coordinates, and Fortran index ordering in packages such as Matlab
- Indexing into a raw array is fast, and common in lower-level applications, so it would be useful to implement the more intuitive XYZ ordering at this level rather than via interpolators (see below)
- Standardizing to one index ordering (XYZ) would mean users would not have to think about the arrangement of the image in memory
Against:

- C index ordering is more familiar to C users
- C index ordering is the default in numpy
- XYZ ordering can be implemented by wrapping by an interpolator

Note that there is no performance penalty for either array ordering, as this is dealt with internally by NumPy. For example, imagine the following:

```python
arr = np.empty((100,50))  # Indexing is C by default
arr2 = arr.transpose()    # Now it is fortran
# There should be no effective difference in speed for the next two lines
b = arr[0]                # get first row of data - most discontiguous memory
b = arr2[:,0]             # gets same data, again most discontiguous memory
```

**Potential problems for fortran ordering**

**Clash between default ordering of numpy arrays and nipy images**

C index ordering is the default in numpy, and using fortran ordering for images might be confusing in some circumstances. Consider for example:

```python
img_obj = load_image('my.img')  # Where the Image class has been changed to implement Fortran ordering
first_z_slice = img_obj[...,0]  # returns a Z slice

img_arr = memmap('my.img', dtype=float32)  # C ordering, the numpy default
img_obj = Image.from_array(img_arr)  # this call may not be correct
first_z_slice = img_obj[...,0]  # in fact returns an X slice
```

I suppose that we could check that arrays are fortran index ordered in the Image __init__ routine.

### 15.2.4 An alternative proposal - XYZ ordering of output coordinates

JT: Another thought, that is a compromise between the XYZ coordinates and Fortran ordering.

To me, having worked mostly with C-type arrays, when I index an array I think in C terms. But, the Image objects have the “warp” attached to them, which describes the output coordinates. We could insist that the output coordinates are XYZT (or make this an option). So, for instance, if the 4x4 transform was the identity, the following two calls would give something like:

```python
>>> interp = interpolator(img)
>>> img[3,4,5] == interp(5,4,3)
True
```

This way, users would be sure in the interpolator of the order of the coordinates, but users who want access to the array would know that they would be using the array order on disk...

I see that a lot of users will want to think of the first coordinate as “x”, but depending on the sampling the [0] slice of img may be the leftmost or the rightmost. To find out which is which, users will have to look at the 4x4 transform (or equivalently the start and the step). So just knowing the first array coordinate is the “x” coordinate still misses some information, all of which is contained in the transform.

MB replied:

I agree that the output coordinates are very important - and I think we all agree that this should be XYZ(T)?
For the raw array indices - it is very common for people to want to do things to the raw image array - the quickstart examples containing a few - and you usually don’t care about which end of X is left in that situation, only which spatial etc dimension the index refers to.

15.3 Registration API Design

This contains design ideas for the end-user api when registering images in nipy.

We want to provide a simple api, but with enough flexibility to allow users to changes various components of the pipeline. We will also provide various Standard scripts that perform typical pipelines.

The pluggable script:

```python
func_img = load_image(filename)
anat_img = load_image(filename)
interpolator = SplineInterpolator(order=3)
metric = NormalizedMutualInformation()
optimizer = Powell()
strategy = RegistrationStrategy(interpolator, metric, optimizer)
w2w = strategy.apply(img_fixed, img_moving)
```

To apply the transform and resample the image:

```python
new_img = resample(img_moving, w2w, interp=interpolator)
```

Or:

```python
new_img = Image(img_moving, w2w*img_moving.coordmap)
```

15.3.1 Transform Multiplication

The multiplication order is important and coordinate systems must make sense. The output coordinates of the mapping on the right-hand of the operator, must match the input coordinates of the mapping on the left-hand side of the operator.

For example, imageA has a mapping from voxels-to-world (v2w), imageB has a mapping from world-to-world (w2w). So the output of imageA, world, maps to the input of imageB, world. We would compose a new mapping (transform) from these mappings like this:

```python
new_coordmap = imageB.coordmap * imageA.coordmap
```

If one tried to compose a mapping in the other order, an error should be raised as the code would detect a mismatch of trying to map output coordinates from imageB, world to the input coordinates of imageA, voxels:

```python
new_coordmap = imageA.coordmap * imageB.coordmap
raise ValueError!!!
```

Note: We should consider a meaningful error message to help people quickly correct this mistake.

One way to remember this ordering is to think of composing functions. If these were functions, the output of the first function to evaluate (imageA.coordmap) is passed as input to the second function (imageB.coordmap). And therefore they must match:

```python
new_coordmap = imageB.coordmap(imageA.coordmap())
```
15.3.2 Matching Coordinate Systems

We need to make sure we can detect mismatched coordinate mappings. The CoordinateSystem class has a check for equality (\_\_eq\_\_ method) based on the axis and name attributes. Long-term this may not be robust enough, but it's a starting place. We should write tests for failing cases of this, if they don't already exists.

15.3.3 CoordinateMap

Recall the CoordinateMap defines a mapping between two coordinate systems, an input coordinate system and an output coordinate system. One example of this would be a mapping from voxel space to scanner space. In a Nifti1 header we would have an affine transform to apply this mapping. The input coordinates would be voxel space, the output coordinates would be world space, and the affine transform provides the mapping between them.

15.4 Repository design

See also Repository API and Can NIPY get something interesting from BrainVISA databases?

For the NIPY system, there seems to be interest for the following:

- Easy distributed computing
- Easy scripting, replicating the same analysis on different data
- Flexibility - easy of inter-operation with other brain imaging systems

At a minimum, this seems to entail the following requirements for the NIPY repository system:

- Unique identifiers of data, which can be abstracted from the most local or convenient data storage
- A mechanism for mapping the canonical data model(s) from NIPY to an arbitrary, and potentially even inconsistent repository structure
- A set of semantic primitives / metadata slots, enabling for example:
  - “all scans from this subject”
  - “the first scan from every subject in the control group”
  - “V1 localizer scans from all subjects”
  - “Extract the average timecourse for each subject from the ROI defined by all voxels with t > 0.005 in the V1 localizer scan for that subject”

These problems are not unique to the problem of brain imaging data, and in many cases have been treated in the domains of database design, geospatial and space telescope data, and the semantic web. Technologies of particular interest include:

- HDF5 - the basis of MINC 2.0 (and potentially NIFTII 2), the most recent development in the more general CDF / HDF series (and very highly regarded). There are excellent python binding available in PyTables.
- Relational database design - it would be nice to efficiently select data based on any arbitrary subset of attributes associated with that data.
- The notion of URI developed under the guidance of the w3c. Briefly, a URI consists of:
  - An authority (i.e. a domain name controlled by a particular entity)
  - A path - a particular resource specified by that authority
  - Abstraction from storage (as opposed to a URL) - a URI does not necessarily include the information necessary for retrieving the data referred to, though it may.
• Ways of dealing with hierarchical data as developed in the XML field (though these strategies could be implemented potentially in other hierarchical data formats - even filesystems).

Note that incorporation of any of the above ideas does not require the use of the actual technology referenced. For example, relational queries can be made in PyTables in many cases more efficiently than in a relational database by storing everything in a single denormalized table. This data structure tends to be more efficient than the equivalent normalized relational database format in the cases where a single data field is much larger than the others (as is the case with the data array in brain imaging data). That said, adherence to standards allows us to leverage existing code which may be tuned to a degree that would be beyond the scope of this project (for example, fast XPath query libraries, as made available via lxml in Python).

15.5 Can NIPY get something interesting from BrainVISA databases?

I wrote this document to try to give more information to the NIPY developers about the present and future of BrainVISA database system. I hope it will serve the discussion opened by Jarrod Millman about a possible collaboration between the two projects on this topic. Unfortunately, I do not know other projects providing similar features (such as BIRN) so I will only focus on BrainVISA.

Yann Cointepas
2006-11-21

15.5.1 Introduction

In BrainVISA, all the database system is home made and written in Python. This system is based on the file system and allows to do requests for both reading and writing (get the name of non existing files). We will change this in the future by defining an API (such the one introduced below) and by using at least two implementations, one relying on a relational database system and one compatible with the actual database system. Having one single API will make it possible, for instance, to work on huge databases located on servers and on smaller databases located in a laptop directory (with some synchronization features). This system will be independent from the BrainVISA application, it could be packaged separately. Unfortunately, we cannot say when this work will be done (our developments are slowed because all our lab will move in a new institute in January 2007). Here is a summary describing actual BrainVISA database system and some thoughts of what it may become.

15.5.2 What is a database in BrainVISA today?

A directory is a BrainVISA database if the structure of its sub-directories and the file names in this directory respect a set of rules. These rules make it possible to BrainVISA to scan the whole directory contents and to identify without ambiguity the database elements. These elements are composed of the following information:

• **Data type**: identify the contents of a data (image, mesh, functional image, anatomical RM, etc). The data types are organized in hierarchy making it possible to decline a generic type in several specialized types. For example, there is a 4D Image type which is specialized in 3D Image. 3D Image is itself declined in several types of which T1 MRI and Brain mask.

• **File format**: Represent the format of files used to record a data. BrainVISA is able to recognize several file formats (for example DICOM, Analyze/SPM, GIS, etc). It is easy to add new data formats and to provide converters to make it possible for existing processes to use these new formats.

• **Files**: contains the names of the files (and/or directories) used to record the data.

• **Attributes**: an attribute is an association between a name and a value. A set of attributes is associated with each element of BrainVISA database. This set represents all of the characteristics of a data (as the image
size, the name of the protocol corresponding to the data or the acquisition parameters). Attributes values
are set by BrainVISA during directory scanning (typically protocol, group, subject, etc.).

It is possible to completely define the set of rules used to convert a directory in a BrainVISA database. That allows the
use of BrainVISA without having to modify an existing file organization. However, the writing of such a system of
rules requires very good knowledge of BrainVISA. This is why BrainVISA is provided with a default data organization
system that can be used easily.

A database can be used for deciding where to write data. The set of rules is used to generate the appropriate file
name according to the data type, file format and attributes. This is a key feature that greatly helps the users and allow
automation.

It is not mandatory to use a database to process data with BrainVISA. However, some important features are not
available when you are using data which are not in a database. For example, the BrainVISA ability to construct a
default output file name when an input data is selected in a process relies on the database system. Moreover, some
processes use the database system to find data; for example, the brain mask viewer tries to find the T1 MRI used to
build the brain mask in order to superimpose both images in an Anatomist window.

15.5.3 A few thoughts about a possible API for repositories

I think the most important point for data repositories is to define an user API. This API should be independent of data
storage and of data organization. Data organization is important because it is very difficult to find a single organization
that covers the needs of all users in the long term. In this API, each data item should have an unique identifier (let’s
call it an URL). The rest of the API could be divided in two parts:

1. An indexation system managing data organization. It defines properties attached to data items (for instance,
   “group” or “subject” can be seen as properties of an FMRI image) as well as possible user requests on the data.
   This indexation API could have several implementations (relational database, BIRN, BrainVISA, etc.).

2. A data storage system managing the link between the URL of a data item and its representation on a local file
   system. This system should take into account various file formats and various file storage systems (e.g. on a
   local file system, on a distant ftp site, as bytes blocks in a relational database).

This separation between indexation and storage is important for the design of databases, it makes it possible, for
instance, to use distant or local data storage, or to define several indexations (i.e. several data organizations) for the
same data. However indexation and data storage are not always independent. For example, they are independent
if we use a relational database for indexation and URLs for storage, but they are not if file or directory names give
indexation information (like in BrainVISA databases described above). At the user level, things can be simpler because
the separation can be hidden in one object: the repository. A repository is composed of one indexation system and one
data storage system and manage all the links between them. The user can send requests to the repository and receive
a set of data items. Each data item contains indexation information (via the indexation system) and gives access to the
data (via the storage system). Here is a sample of what-user-code-could-be to illustrate what I have in mind followed
by a few comments:

```python
# Get an acces to one repository
repository = openRepository( repositoryURL )
# Create a request for selection of all the FMRI in the repository
request = 'SELECT * FROM FMRI'
# Iterate on data items in the repository
for item in repository.select( request ):
    print item.url
    # Item is a directory-like structure for properties access
    for property in item:
        print property, '=', item[ property ]
    # Retrieve the file(s) (and directory(s) if any) from the data storage system
    # and convert it to NIFTI format (if necessary).
    files = item.getLocalFiles( format='NIFTI' )
```

118 Chapter 15. Code discussions
niftiFileName = files[ 0 ]

# Read the image and do something with it
...

1. I do not yet have a good idea of how to represent requests. Here, I chose to use SQL since it is simple to understand.

2. This code does not make any assumption on the properties that are associated to an FMRI image.

3. The method getLocalFiles can do nothing more than return a file name if the data item correspond to a local file in NIFTI format. But the same code can be used to acces a DICOM image located in a distant ftp server. In this case, getLocalFiles will manage the transfer of the DICOM file, then the conversion to the required NIFTI format and return name of temporary file(s).

4. getLocalFiles cannot always return just one file name because on the long term, there will be many data types (FMRI, diffusion MRI, EEG, MEG, etc.) that are going to be stored in the repositories. These different data will use various file formats. Some of these formats can use a combination of files and directories (for instance, CTF MEG raw data are stored in a directory (*.ds), the structural sulci format of BrainVISA is composed of a file(*.arg) and a directory (*.data), NIFTI images can be in one or two files, etc.).

5. The same kind of API can be used for writing data items in a repository. One could build a data item, adds properties and files and call something like repository.update( item ).

15.6 Repository API

See also Repository design and Can NIPY get something interesting from BrainVISA databases?

FMRI datasets often have the structure:

- Group (sometimes) e.g. Patients, Controls
  - Subject e.g. Subject1, Subject2
    - Session e.g. Sess1, Sess1

How about an interface like:

```python
repo = GSSRespository(
    root_dir = '/home/me/data/experiment1',
    groups = {'patients':
        {'subjects':
            {'patient1':
                {'sess1':
                    'filter': 'raw*nii'},
                {'sess2':
                    'filter': 'raw*nii'}
            },
            {'patient2':
                {'sess1':
                    'filter': 'raw*nii'}
            }
        },
        'controls':
        {'subjects':
            {'controll':
                {'sess1':
                    'filter': 'raw*nii'}
            }
        }
    }
)
```
{'sess2':
   {'filter': 'raw*nii'}
},
{'control2':
   {'sess1':
      {'filter': 'raw*nii'}
   }
}
}

for group in repo.groups:
    for subject in group.subjects:
        for session in subject.sessions:
            img = session.image
            # do something with image

We would need to think about adding metadata such as behavioral data from the scanning session, and so on. I suppose this will help us move transparently to using something like HDF5 for data storage.

15.7 What would pipelining look like?

Imagine a repository that is a modified version of the one in Repository API

Then:

my_repo = SubjectRepository('/some/structured/file/system')
my_designmaker = MyDesignParser()  # Takes parameters from subject to create design
my_pipeline = Pipeline(
    realignerfactory('fsl'),
    slicetimerfactory('nipy', 'linear'),
    coregisterfactory('fsl', 'flirt'),
    normalizerfactory('spm'),
    filterfactory('nipy', 'smooth', 8),
    designfactory('nipy', my_designmaker),
)

my_analysis = SubjectAnalysis(my_repo, subject_pipeline=my_pipeline)
my_analysis.do()
my_analysis.archive()

15.8 Simple image viewer

15.9 Other attempts

15.10 Defining use cases

15.10.1 Transformation use cases

Use cases for defining and using transforms on images.

We should be very careful to only use the terms x, y, z to refer to physical space. For voxels, we should use i, j, k, or i’, j’, k’ (i prime, j prime k prime).

I have an image \( \text{Img} \).

**Image Orientation**

I would like to know what the voxel sizes are.

I would like to determine whether it was acquired axially, coronally or sagittally. What is the brain orientation in relation to the voxels? Has it been acquired at an oblique angle? What are the voxel dimensions?:

```python
img = load_image(file)
cm = img.coordmap
print cm

input_coords
  axis_i:
  axis_j:
  axis_k:

effective pixel dimensions
  axis_i: 4mm
  axis_j: 2mm
  axis_k: 2mm

input/output mapping
  <Affine Matrix>
```

```plaintext
<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>90</td>
<td>90</td>
<td>0</td>
</tr>
<tr>
<td>j</td>
<td>90</td>
<td>0</td>
<td>90</td>
</tr>
<tr>
<td>k</td>
<td>180</td>
<td>90</td>
<td>90</td>
</tr>
</tbody>
</table>
```

input axis_i maps exactly to output axis_z
input axis_j maps exactly to output axis_y
input axis_k maps exactly to output axis_x flipped 180

output_coords
  axis0: Left -> Right
  axis1: Posterior -> Anterior
  axis2: Inferior -> Superior

In the case of a mapping that does not exactly align the input and output axes, something like:

```python
... input/output mapping
  <Affine Matrix>
```

input axis0 maps closest to output axis2

15.10. Defining use cases
input axis1 maps closest to output axis1
input axis2 maps closest to output axis0

... If the best matching axis is reversed compared to input axis:
... input axis0 maps [closest|exactly] to negative output axis2

and so on.

Creating transformations / co-ordinate maps

I have an array `pixelarray` that represents voxels in an image and have a matrix/transform `mat` which represents the relation between the voxel coordinates and the coordinates in scanner space (world coordinates). I want to associate the array with the matrix:

```python
img = load_image(infile)
pixelarray = np.asarray(img)
```

(`pixelarray` is an array and does not have a coordinate map.):

```python
pixelarray.shape
(40, 256, 256)
```

So, now I have some arbitrary transformation matrix:

```python
mat = np.zeros((4, 4))
mat[0, 2] = 2 # giving x mm scaling
mat[1, 1] = 2 # giving y mm scaling
mat[2, 0] = 4 # giving z mm scaling
mat[3, 3] = 1 # because it must be so
# Note inverse diagonal for zyx->xyz coordinate flip
```

I want to make an `Image` with these two:

```python
coordmap = voxel2mm(pixelarray.shape, mat)
img = Image(pixelarray, coordmap)
```

The `voxel2mm` function allows separation of the image `array` from the size of the array, e.g.:

```python
coordmap = voxel2mm((40, 256, 256), mat)
```

We could have another way of constructing image which allows passing of `mat` directly:

```python
img = Image(pixelarray, mat=mat)
```

or:

```python
img = Image.from_data_and_mat(pixelarray, mat)
```

but there should be “only one (obvious) way to do it”.

Composing transforms

I have two images, `img1` and `img2`. Each image has a voxel-to-world transform associated with it. (The “world” for these two transforms could be similar or even identical in the case of an fmri series.) I would like to get from voxel coordinates in `img1` to voxel coordinates in `img2`, for resampling:
Neuroimaging in Python Documentation, Release 0.3.0

```python
imgA = load_image(infile_A)
vx2mmA = imgA.coordmap
imgB = load_image(infile_B)
vx2mmB = imgB.coordmap
mm2vxB = vx2mmB.inverse

# I want to first apply transform implied in cmA, then the inverse of transform implied in cmB. If these are matrices then this would be
# np.dot(mm2vxB, vx2mmA)
voxA_to_voxB = mm2vxB.composewith(vx2mmA)

The (matrix) multiply version of this syntax would be:

voxA_to_voxB = mm2vxB * vx2mmA

Composition should be of form Second.composewith(First) - as in voxA_to_voxB = mm2vxB.composewith(vx2mmA) above. The alternative is First.composewith(Second), as in voxA_to_voxB = vx2mmA.composewith(mm2vxB). We choose Second.composewith(First) on the basis that people need to understand the mathematics of function composition to some degree - see wikipedia_function_composition.

Real world to real world transform

We remind each other that a mapping is a function (callable) that takes coordinates as input and returns coordinates as output. So, if \( M \) is a mapping then:

\[
[i', j', k'] = M(i, j, k)
\]

where the \( i, j, k \) tuple is a coordinate, and the \( i', j', k' \) tuple is a transformed coordinate.

Let us imagine we have somehow come by a mapping \( T \) that relates a coordinate in a world space (mm) to other coordinates in a world space. A registration may return such a real-world to real-world mapping. Let us say that \( V \) is a useful mapping matching the voxel coordinates in \( img1 \) to voxel coordinates in \( img2 \). If \( img1 \) has a voxel to mm mapping \( M1 \) and \( img2 \) has a mm to voxel mapping of \( inv_M2 \), as in the previous example (repeated here):

```python
imgA = load_image(infile_A)
vx2mmA = imgA.coordmap
imgB = load_image(infile_B)
vx2mmB = imgB.coordmap
mm2vxB = vx2mmB.inverse

then the registration may return the some coordinate map, \( T \) such that the intended mapping \( V \) from voxels in \( img1 \) to voxels in \( img2 \) is:

```python
mm2vxB_map = mm2vxB.mapping
vx2mmA_map = vx2mmA.mapping
V = mm2vxB_map.composewith(T.composedwith(vx2mmA_map))

To support this, there should be a CoordinateMap constructor that looks like this:

\[
T_{\text{coordmap}} = \text{mm2mm}(T)
\]

where \( T \) is a mapping, so that:

\[
V_{\text{coordmap}} = \text{mm2vxb.composewith}(T_{\text{coordmap}.\text{composewith}(vx2mmA)})
\]

I have done a coregistration between two images, \( img1 \) and \( img2 \). This has given me a voxel-to-voxel transformation and I want to store this transformation in such a way that I can use this transform to resample \( img1 \) to \( img2 \). Resampling use cases

15.10. Defining use cases
I have done a coregistration between two images, \textit{img1} and \textit{img2}. I may want this to give me a worldA-to-worldB transformation, where worldA is the world of voxel-to-world for \textit{img1}, and worldB is the world of voxel-to-world of \textit{img2}.

My \textit{img1} has a voxel to world transformation. This transformation may (for example) have come from the scanner that acquired the image - so telling me how the voxel positions in \textit{img1} correspond to physical coordinates in terms of the magnet isocenter and millimeters in terms of the primary gradient orientations (x, y and z). I have the same for \textit{img2}. For example, I might choose to display this image resampled so each voxel is a 1mm cube.

Now I have these transformations: ST(\textit{img1}-V2W), and ST(\textit{img2}-V2W) (where ST is \textit{scanner transform} as above, and V2W is voxel to world).

I have now done a coregistration between \textit{img1} and \textit{img2} (somehow) - giving me, in addition to \textit{img1} and \textit{img2}, a transformation that registers \textit{img1} and \textit{img2}. Let’s call this transformation V2V(\textit{img1}, \textit{img2}), where V2V is voxel-to-voxel.

In actuality \textit{img2} can be an array of images, such as series of fMRI images and I want to align all the \textit{img2} series to \textit{img1} and then take these voxel-to-voxel aligned images (the \textit{img1} and \textit{img2} array) and remap them to the world space (voxel-to-world). Since remapping is an interpolation operation I can generate errors in the resampled pixel values. If I do more than one resampling, error will accumulate. I want to do only a single resampling. To avoid the errors associated with resampling I will build a \textit{composite transformation} that will chain the separate voxel-to-voxel and voxel-to-world transformations into a single transformation function (such as an affine matrix that is the result of multiplying the several affine matrices together). With this single \textit{composite transformation} I now resample \textit{img1} and \textit{img2} and put them into the world coordinate system from which I can make measurements.

15.10.2 Image model use cases

In which we lay out the various things that users and developers may want to do to images. See also \textit{Resampling use cases}

**Taking a mean over a 4D image**

We could do this much more simply than below, this is just an example of reducing over a particular axis:

```python
# take mean of 4D image
from glob import glob
import numpy as np
import nipy as ni

fname = 'some4d.nii'

img_list = ni.load_list(fname, axis=3)
vol0 = img_list[0]
arr = vol0.array[:]
for vol in img_list[1:]:
    arr += vol.array
mean_img = ni.Image(arr, vol0.coordmap)
ni.save(mean_img, 'mean_some4d.nii')
```

**Taking mean over series of 3D images**

Just to show how this works with a list of images:
# take mean of some PCA volumes
fnames = glob('some3d*.nii')
vol0 = ni.load(fnames[0])
arr = vol0.array[:]
for fname in fnames[1:]:
    vol = ni.load(fname)
    arr += vol.array
mean_img = ni.Image(arr, vol0.coordmap)
ni.save(mean_img, 'mean_some3ds.nii')

Simple motion correction

This is an example of how the 4D -> list of 3D interface works:

## motion correction
img_list = ni.load_list(fname, axis=3)
reggie = ni.interfaces.fsl.Register(tol=0.1)
vol0 = img_list[0]
mocod = [] # unresliced
rmocod = [] # resliced
for vol in img_list[1:]:
    rcoord_map = reggie.run(moving=vol, fixed=vol0)
    cmap = ni.ref.compose(rcoord_map, vol0.coordmap)
mocovol = ni.Image(vol.array, cmap)
    # But...
    try:
        a_vol = ni.Image(vol.array, rcoord_map)
        except CoordmapError, msg
            assert msg == 'need coordmap with voxel input'
mocod.append(mocovol)
rmocodvol = ni.reslice(mocovol, vol0)
rmocod.append(rmocodvol)
rmocod_img = ni.list_to_image(rmocodvol)
ni.save(rmocod_img, 'rsome4d.nii')
try:
    mocod_img = ni.list_to_image(mocovol)
except ImageListError:
    print 'That is what I thought; the transforms were not the same'

Slice timing

Here putting 3D image into an image list, and back into a 4D image / array:

## slice timing
img_list = ni.load_list(fname, axis=2)
slicetimer = ni.interfaces.fsl.SliceTime(algorithm='linear')
vol0 = img_list[0]
try:
    vol0.timestamp
except AttributeError:
    print 'we do not have a timestamp'
try:
    vol0.slicetimes
except AttributeError:
    print 'we do not have slicetimes'
try:
Creating an image given data and affine

Showing how we would like the image creation API to look:

```python
# making an image from an affine
data = img.array
affine = np.eye(4)
scanner_img = ni.Image(data, ni.ref.voxel2scanner(affine))
mni_img = ni.Image(data, ni.ref.voxel2mni(affine))
```

Coregistration / normalization

Demonstrating coordinate maps and non-linear resampling:

```python
# coregistration and normalization
anat_img = ni.load_image('anatomical.nii')
func_img = ni.load_image('epi4d.nii')
template = ni.load_image('mni152T1.nii')

# coreg
coreger = ni.interfaces.fsl.flirt(tol=0.2)
coreg_cmap = coreger.run(fixed=func_img, moving=anat_img)
c_anat_img = ni.Image(anat_img.data, coreg_cmap.compose_with(anat_img.cmap))

# calculate normalization parameters
template_cmap = template.coordmap
template_dims = template.data.shape
c_anat_cmap = c_anat_img.coordmap
```
normalizer = ni.interfaces.fsl.fnirt(param=3)
norm_cmap = normalizer.run(moving=template, fixed=c_anat_img)

# resample anatomical using calculated coordinate map
full_cmap = norm_cmap.composed_with(template_cmap)
w_anat_data = img.resliced_to_grid(full_cmap, template_dims)
w_anat_img = ni.Image(w_anat_data, template.coordmap)

# resample functionals with calculated coordinate map
w_func_list = []
for img in ni.image_list(func_img, axis=3):
    w_img_data = img.resliced_to_grid(full_cmap, template_dims)
w_func_list.append(ni.Image(w_img_data, template.coordmap))
ni.save(ni.list_to_image(w_func_list), 'stsome4d.nii')

15.10.3 Resampling use cases

Use cases for image resampling. See also images.

15.10.4 Batching use cases

Using the nipy framework for creating scripts to process whole datasets, for example movement correction, coregistration of functional to structural (intermodality), smoothing, statistics, inference.

15.11 Defining use cases

15.11.1 Refactoring imagelists

Use cases for ImageList

Thus far only used in anger in nipy.modalities.fmri.fmristat.model, similarly in nipy.modalities.fmri.spm.model.

From that file, an object obj of class FmriImageList must:

- return 4D array from np.asarray(obj), such that the first axis (axis 0) is the axis over which the model is applied
- be indexable such that obj[0] returns an Image instance, with valid shape and coordmap attributes for a time-point 3D volume in the 4D time-series.
- have an attribute volume_start_times giving times of the start of each of the volumes in the 4D time series.
- Return the number of volumes in the time-series from len(obj)
15.12 Software Design

15.12.1 VTK datasets

Here we describe the VTK dataset model, because of some parallels with our own idea of an image object. The document is from the VTK book - [VTK4]

See also:
- http://www.vtk.org/Wiki/VTK/Writing_VTK_files_using_python
- http://public.kitware.com/cgi-bin/viewcvs.cgi/checkout/Examples/DataManipulation/Python/BuildUGrid.py?root=VTK&content-type=text/plain

What is a VTK dataset?

VTK datasets represent discrete spatial data.

Datasets consist of two components:

- *organizing structure* - the topology and geometry
- *data attributes* - data that can be attached to the topology / geometry above.

Structure: topology / geometry

The structure part of a dataset is the part that gives the position and connection of points in 3D space.

Let us first import `vtk` for our code examples.

```python
>>> import vtk
```

An id is an index into a given vector

We introduce *id* to explain the code below. An id is simply an index into a vector, and is therefore an integer. Of course the id identifies the element in the vector; as long as you know which vector the id refers to, you can identify the element.

```python
>>> pts = vtk.vtkPoints()
>>> id = pts.InsertNextPoint(0, 0, 0)
>>> id == 0
True
>>> id = pts.InsertNextPoint(0, 1, 0)
>>> id == 1
True
>>> pts.GetPoint(1) == (0.0, 1.0, 0.0)
True
```
A dataset has one or more points Points have coordinates in 3 dimensions, in the order x, y, z - see http://www.vtk.org/doc/release/5.4/html/a00374.html - GetPoint()

```python
>>> pts = vtk.vtkPoints()
>>> pts.InsertNextPoint(0, 0)  # needs 3 coordinates
Traceback (most recent call last):
 ...
TypeError: function takes exactly 3 arguments (2 given)
```  
```python
>>> _ = pts.InsertNextPoint(0, 0, 0)  # returns point index in point array
>>> pts.GetPoint(0)
(0.0, 0.0, 0.0)
```  
```python
>>> _ = pts.InsertNextPoint(0, 1, 0)
>>> _ = pts.InsertNextPoint(0, 0, 1)
```  
A dataset has one or more cells A cell is a local specification of the connection between points - an atom of topology in VTK. A cell has a type, and a list of point ids. The point type determines (by convention) what the connectivity of the list of points should be. For example we can make a cell of type `vtkTriangle`. The first point starts the triangle, the next point is the next point in the triangle counterclockwise, connected to the first and third, and the third is the remaining point, connected to the first and second.

```python
>>> VTK_TRIANGLE = 5  # A VTK constant identifying the triangle type
>>> triangle = vtk.vtkTriangle()
>>> isinstance(triangle, vtk.vtkCell)
True
>>> triangle.GetCellType() == VTK_TRIANGLE
True
>>> pt_ids = triangle.GetPointIds()  # these are default (zeros) at the moment
>>> [pt_ids.GetId(i) for i in range(pt_ids.GetNumberOfIds())] == [0, 0, 0]
True
```  
Here we set the ids. The ids refer to the points above. The system does not know this yet, but it will because, later, we are going to associate this cell with the points, in a dataset object.

```python
>>> for i in range(pt_ids.GetNumberOfIds()): pt_ids.SetId(i, i)
```  
Associating points and cells We make the most general possible of VTK datasets - the unstructured grid.

```python
>>> ugrid = vtk.vtkUnstructuredGrid()
>>> ugrid.Allocate(1, 1)
>>> ugrid.SetPoints(pts)
>>> id = ugrid.InsertNextCell(VTK_TRIANGLE, pt_ids)
```  
Data attributes

So far we have specified a triangle, with 3 points, but no associated data.

You can associate data with cells, or with points, or both. Point data associates values (e.g. scalars) with the points in the dataset. Cell data associates values (e.g. scalars) with the cells - in this case one (e.g) scalar value with the whole triangle.

```python
>>> pt_data = ugrid.GetPointData()
>>> cell_data = ugrid.GetCellData()
```  
There are many data attributes that can be set, include scalars, vectors, normals (normalized vectors), texture coordinates and tensors, using (respectively) `{pt|cell|_data}.Get|Set}{Scalars|Vectors|Normals|TCoords|Tensors}`. For example:
>>> pt_data.GetScalars() is None
True

But we can set the scalar (or other) data:

```python
>>> tri_pt_data = vtk.vtkFloatArray()
>>> for i in range(3): _ = tri_pt_data.InsertNextValue(i)
>>> _ = pt_data.SetScalars(tri_pt_data)
```

To the cells as well, or instead, if we want. Don’t forget there is only one cell.

```python
>>> tri_cell_data = vtk.vtkFloatArray()
>>> _ = tri_cell_data.InsertNextValue(3)
>>> _ = cell_data.SetScalars(tri_cell_data)
```

You can set different types of data into the same dataset:

```python
>>> tri_pt_vecs = vtk.vtkFloatArray()
>>> tri_pt_vecs.SetNumberOfComponents(3)
>>> tri_pt_vecs.InsertNextTuple3(1, 1, 1)
>>> tri_pt_vecs.InsertNextTuple3(2, 2, 2)
>>> tri_pt_vecs.InsertNextTuple3(3, 3, 3)
>>> _ = pt_data.SetVectors(tri_pt_vecs)
```

If you want to look at what you have, run this code

```python
# .. testcode:: when live
# make a dataset mapper and actor for our unstructured grid
mapper = vtk.vtkDataSetMapper()
mapper.SetInput(ugrid)
actor = vtk.vtkActor()
actor.SetMapper(mapper)
# Create the usual rendering stuff.
ren = vtk.vtkRenderer()
renWin = vtk.vtkRenderWindow()
renWin.AddRenderer(ren)
iren = vtk.vtkRenderWindowInteractor()
iren.SetRenderWindow(renWin)
# add the actor
ren.AddActor(actor)
# Render the scene and start interaction.
iren.Initialize()
renWin.Render()
iren.Start()
```
16.1 Tricked out emacs for python coding

Various ways to configure your emacs that you might find useful.
See emacs_python_mode for a good summary.

16.1.1 ReST mode

For editing ReST documents like this one. You may need a recent version of the rst.el file from the docutils site.

rst mode automates many important ReST tasks like building and updating table-of-contents, and promoting or demoting section headings. Here is the basic .emacs configuration:

```
(require 'rst)
(setq auto-mode-alist
  (append '(
               (\.*\.[tT][xX][tT]$ . rst-mode)
               (\.*\.[rR][sS][tT]$ . rst-mode)
               (\.*\.[rR][eE][sS][tT]$ . rst-mode)
         ) auto-mode-alist))
```

Some helpful functions:

C-c TAB - rst-toc-insert

Insert table of contents at point

C-c C-u - rst-toc-update

Update the table of contents at point

C-c C-l rst-shift-region-left

Shift region to the left

C-c C-r rst-shift-region-right

Shift region to the right

Note: On older Debian-based releases, the default M-x rst-compile command uses rst2html.py whereas the command installed is rst2html. Symlink was required as a quick fix.
16.1.2 doctest mode

This useful mode for writing doctests (doctest-mode.el) comes with python-mode package on Debian-based systems. Otherwise see doctest-mode project page.

16.1.3 code checkers

Code checkers within emacs can be useful to check code for errors, unused variables, imports and so on. Alternatives are pychecker, pylint and pyflakes. Note that rope (below) also does some code checking. pylint and pyflakes work best with emacs flymake, which usually comes with emacs.

pychecker

This appears to be plumbed in with python-mode, just do M-x py-pychecker-run. If you try this, and pychecker is not installed, you will get an error. You can install it using your package manager (pychecker on Debian-based systems) or from the pychecker webpage.

pylint

Install pylint. Debian packages pylint as pylint. Put the flymake .emacs snippet in your .emacs file. You will see, in the emacs_python_mode page, that you will need to save this:

```python
#!/usr/bin/env python
import re
import sys
from subprocess import *
p = Popen("pylint -f parseable -r n --disable-msg-cat=C,R %s" % sys.argv[1], shell = True, stdout = PIPE).stdout
for line in p.readlines():
    match = re.search("\[(\[WE\])(, (.+?)?\])", line)
    if match:
        kind = match.group(1)
        func = match.group(3)

        if kind == "W":
            msg = "Warning"
        else:
            msg = "Error"

        if func:
            line = re.sub("\[(\[WE\])(, (.+?)?\])", "%s (%s):" % (msg, func), line)
        else:
            line = re.sub("\[(\[WE\])?\]", "%s: " % msg, line)
    print line,
p.close()
```

as epylint somewhere on your system path, and test that epylint somepyfile.py works.
pyflakes


flymake.emacs snippet

Add this to your .emacs file:

```lisp
;; code checking via flymake
;; set code checker here from "epylint", "pyflakes"
(setq pycodechecker "pyflakes")
(when (load "flymake" t)
  (defun flymake-pycodecheck-init ()
    (let* ((temp-file (flymake-init-create-temp-buffer-copy
                       'flymake-create-temp-inplace))
           (local-file (file-relative-name
temp-file
                      (file-name-directory buffer-file-name))))
      (list pycodechecker (list local-file)))
    (add-to-list 'flymake-allowed-file-name-masks
                 '("\.py\"" flymake-pycodecheck-init)))))
```

and set which of pylintrc (“epylint”) or pyflakes (“pyflakes”) you want to use.

You may also consider using the flymake-cursor functions, see the pyflakes section of the emacs_python_mode page for details.

16.1.4 ropemacs

rope is a python refactoring library, and ropemacs is an emacs interface to it, that uses pymacs. pymacs is an interface between emacs lisp and python that allows emacs to call into python and python to call back into emacs.

Install

- rope - by downloading from the link, and running python setup.py install in the usual way.
- pymacs - probably via your package manager - for example apt-get install pymacs
- ropemacs - download from link, python setup.py install

You may need to make sure your gnome etc sessions have the correct python path settings - for example settings in .gnomerc as well as the usual .bashrc.

Make sure you can import ropemacs from python (which should drop you into something lispey). Add these lines somewhere in your .emacs file:

```lisp
(require 'pymacs)
(pymacs-load "ropemacs" "rope-"
```

and restart emacs. When you open a python file, you should have a rope menu. Note C-c g - the excellent goto-definition command.
16.1.5 Switching between modes

You may well find it useful to be able to switch fluidly between python mode, doctest mode, ReST mode and flymake mode (pylint). You can attach these modes to function keys in your .emacs file with something like:

(global-set-key [f8] 'flymake-mode)
(global-set-key [f9] 'python-mode)
(global-set-key [f10] 'doctest-mode)
(global-set-key [f11] 'rst-mode)

16.1.6 emacs code browser

Not really python specific, but a rather nice set of windows for browsing code directories, and code - see the ECB page. Again, your package manager may help you (apt-get install ecb).

16.2 Setting up virtualenv

Contents

• Setting up virtualenv
  – Overview
  – Installing
  – Setup virtualenv
  – Create a virtualenv
  – Activate a virtualenv
  – Install packages into a virtualenv
  – Pragmatic virtualenv
  – Installing ETS 3.0.0

16.2.1 Overview

virtualenv is a tool that allows you to install python packages in isolated environments. In this way you can have multiple versions of the same package without interference. I started using this to easily switch between multiple versions of numpy without having to constantly reinstall and update my symlinks. I also did this as a way to install software for Scipy2008, like the Enthought Tool Suite (ETS), in a way that would not effect my current development environment.

This tutorial is based heavily on a blog entry from Prabhu. I’ve extended his shell script to make switching between virtual environments a one-command operation. (Few others who should be credited for encouraging me to use virtualenv: Gael, Jarrod, Fernando)

16.2.2 Installing

Download and install the tarball for virtualenv:

tar xzf virtualenv-1.1.tar.gz
cd virtualenv-1.1
python setup.py install --prefix=$HOME/local

Note: I install in a local directory, your install location may differ.
16.2.3 Setup virtualenv

Setup a base virtualenv directory. I create this in a local directory, you can do this in a place of your choosing. All virtual environments will be installed as subdirectories in here:

```bash
cd ~/local
mkdir -p virtualenv
```

16.2.4 Create a virtualenv

Create a virtual environment. Here I change into my virtualenv directory and create a virtual environment for my numpy-1.1.1 install:

```bash
cd virtualenv/
virtualenv numpy-1.1.1
```

16.2.5 Activate a virtualenv

Set the numpy-1.1.1 as the active virtual environment:

```bash
ln -s numpy-1.1.1/bin/activate .
```

We enable the numpy-1.1.1 virtual environment by sourcing it’s activate script. This will prepend our PATH with the currently active virtual environment:

```bash
# note: still in the ~/local/virtualenv directory
source activate
```

We can see our PATH with the numpy-1.1.1 virtual environment at the beginning. Also not the label of the virtual environment prepends our prompt:

```bash
(numpy-1.1.1)cburns@~ 20:23:54 $ echo $PATH
/Users/cburns/local/virtualenv/numpy-1.1.1/bin:
/Library/Frameworks/Python.framework/Versions/Current/bin:
/Users/cburns/local/bin:
/usr/bin:/usr/sbin:/sbin:/usr/local/bin:/usr/X11/bin:/usr/local/git/bin
```

16.2.6 Install packages into a virtualenv

Then we install numpy-1.1.1 into the virtual environment. In order to install packages in the virtual environment, you need to use the python or easy_install from that virtualenv:

```bash
~/local/virtualenv/numpy-1.1.1/bin/python setup.py install
```

At this point any package I install in this virtual environment will only be used when the environment is active.

16.2.7 Pragmatic virtualenv

There are a few more manual steps in the above process then I wanted, so I extended the shell script that Prabhu wrote to make this a simple one-command operation. One still needs to manually create each virtual environment, and install packages, but this script simplifies activating and deactivating them.

The venv_switch.sh script will:
• Activate the selected virtual environment. (Or issue an error if it doesn’t exist.)
• Launch a new bash shell using the ~/.virtualenvrc file which sources the virtualenv/activate script.
• The activate script modifies the PATH and prepends the bash prompt with the virtualenv label.

venv_switch.sh:

#!/bin/sh
# venv_switch.sh
# switch between different virtual environments

# verify a virtualenv is passed in
if [ $# -ne 1 ]
then
    echo 'Usage: venv_switch venv-label'
    exit -1
fi

# verify the virtualenv exists
VENV_PATH=~/local/virtualenv/$1

# activate env script
ACTIVATE_ENV=~/local/virtualenv/activate

echo $VENV_PATH
if [ -e $VENV_PATH ]
then
    echo 'Switching to virtualenv' $VENV_PATH
    echo "Starting new bash shell. Simply 'exit' to return to previous shell"
else
    echo 'Error: virtualenv' $VENV_PATH 'does not exist!'
    exit -1
fi

rm $ACTIVATE_ENV
ln -s ~/local/virtualenv/$1/bin/activate $ACTIVATE_ENV

# Launch new terminal
bash --rcfile ~/.virtualenvrc

Now to activate our numpy-1.1.1 virtual environment, we simply do:

venv_switch.sh numpy-1.1.1

To deactivate the virtual environment and go back to your original environment, just exit the bash shell:

exit

The rcfile used to source the activate script. I first source my .profile to setup my environment and custom prompt, then source the virtual environment. .virtualenvrc:

# rc file to initialize bash environment for virtualenv sessions

# first source the bash_profile
source ~/.bash_profile

# source the virtualenv
source ~/local/virtualenv/activate
16.2.8 Installing ETS 3.0.0

As another example, I installed ETS 3.0.0 for the Tutorial sessions at Scipy2008. (Note the prerequisites.)

Set up an ets-3.0.0 virtualenv:

cburns@virtualenv 15:23:50 $ pwd
/Users/cburns/local/virtualenv

cburns@virtualenv 15:23:50 $ virtualenv ets-3.0.0
New python executable in ets-3.0.0/bin/python
Installing setuptools............done.

cburns@virtualenv 15:24:29 $ ls
activate ets-3.0.0 numpy-1.1.1 numpy-1.2.0b2

Switch into my ets-3.0.0 virtualenv using the venv_switch.sh script:

cburns@~ 15:29:12 $ venv_switch.sh ets-3.0.0
Switching to virtualenv /Users/cburns/local/virtualenv/ets-3.0.0
Starting new bash shell. Simply ‘exit’ to return to previous shell

Install ETS using easy_install. Note we need to use the easy_install from our ets-3.0.0 virtual environment:

(ets-3.0.0)cburns@~ 15:31:41 $ which easy_install
/Users/cburns/local/virtualenv/ets-3.0.0/bin/easy_install

(ets-3.0.0)cburns@~ 15:31:48 $ easy_install ETS
Part IV

FAQ
17.1 Why nipy?

We are writing NIPY because we hope that it will solve several problems in the field at the moment.

We are concentrating on FMRI analysis, so we’ll put the case for that part of neuroimaging for now.

There are several good FMRI analysis packages already - for example SPM, FSL and AFNI. For each of these you can download the source code.

Like SPM, AFNI and FSL, we think source code is essential for understanding and development.

With these packages you can do many analyses. Some problems are that:

- The packages don’t mix easily. You’ll have to write your own scripts to mix between them; this is time-consuming and error-prone, because you will need good understanding of each package
- Because they don’t mix, researchers usually don’t try and search out the best algorithm for their task - instead they rely on the software that they are used to
- Each package has its own user community, so it’s a little more difficult to share software and ideas
- The core development of each language belongs in a single lab.

Another, more general problem, is planning for the future. We need a platform that can be the basis for large scale shared development. For various reasons, it isn’t obvious to us that any of these three is a good choice for common, shared development. In particular, we think that Python is the obvious choice for a large open-source software project. By comparison, matlab is not sufficiently general or well-designed as a programming language, and C / C++ are too hard and slow for scientific programmers to read or write. See why-python for this argument in more detail.

We started NIPY because we want to be able to:

- support an open collaborative development environment. To do this, we will have to make our code very easy to understand, modify and extend. If make our code available, but we are the only people who write or extend it, in practice, that is closed software.
- make the tools that allow developers to pick up basic building blocks for common tasks such as registration and statistics, and build new tools on top.
- write a scripting interface that allows you to mix in routines from the other packages that you like or that you think are better than the ones we have.
- design ways of interacting with the data and analysis stream that help you organize both. That way you can more easily keep track of your analyses. We also hope this will make analyses easier to run in parallel, and therefore much faster.
17.2 Why python?

The choice of programming language has many scientific and practical consequences. Matlab is an example of a high-level language. Languages are considered high level if they are able to express a large amount of functionality per line of code; other examples of high level languages are Python, Perl, Octave, R and IDL. In contrast, C is a low-level language. Low level languages can achieve higher execution speed, but at the cost of code that is considerably more difficult to read. C++ and Java occupy the middle ground sharing the advantages and the disadvantages of both levels.

Low level languages are a particularly ill-suited for exploratory scientific computing, because they present a high barrier to access by scientists that are not specialist programmers. Low-level code is difficult to read and write, which slows development ([Prechelt2000ECS], [boehm1981], [Walston1977MPM]) and makes it more difficult to understand the implementation of analysis algorithms. Ultimately this makes it less likely that scientists will use these languages for development, as their time for learning a new language or code base is at a premium. Low level languages do not usually offer an interactive command line, making data exploration much more rigid. Finally, applications written in low level languages tend to have more bugs, as bugs per line of code is approximately constant across many languages [brooks78].

In contrast, interpreted, high-level languages tend to have easy-to-read syntax and the native ability to interact with data structures and objects with a wide range of built-in functionality. High level code is designed to be closer to the level of the ideas we are trying to implement, so the developer spends more time thinking about what the code does rather than how to write it. This is particularly important as it is researchers and scientists who will serve as the main developers of scientific analysis software. The fast development time of high-level programs makes it much easier to test new ideas with prototypes. Their interactive nature allows researchers flexible ways to explore their data.

SPM is written in Matlab, which is a high-level language specialized for matrix algebra. Matlab code can be quick to develop and is relatively easy to read. However, Matlab is not suitable as a basis for a large-scale common development environment. The language is proprietary and the source code is not available, so researchers do not have access to core algorithms making bugs in the core very difficult to find and fix. Many scientific developers prefer to write code that can be freely used on any computer and avoid proprietary languages. Matlab has structural deficiencies for large projects: it lacks scalability and is poor at managing complex data structures needed for neuroimaging research. While it has the ability to integrate with other languages (e.g., C/C++ and FORTRAN) this feature is quite impoverished. Furthermore, its memory handling is weak and it lacks pointers - a major problem for dealing with the very large data structures that are often needed in neuroimaging. Matlab is also a poor choice for many applications such as system tasks, database programming, web interaction, and parallel computing. Finally, Matlab has weak GUI tools, which are crucial to researchers for productive interactions with their data.
18.1 How do you spell licence?

If you are British you spell it differently from Americans, sometimes:
http://www.tiscali.co.uk/reference/dictionaries/english/data/d0082350.html
As usual the American spelling rule (always use s) was less painful and arbitrary, so I (MB) went for that.

18.2 Why did you choose BSD?

We have chosen BSD licensing, for compatibility with SciPy, and to increase input from developers in industry. Wherever possible we will keep packages that can have BSD licensing separate from packages needing a GPL license. Our choices were between:

- **BSD**
- **GPL**

John Hunter made the argument for the BSD license in *johns-bsd-pitch*, and we agree. Richard Stallman makes the case for the GPL here: http://www.gnu.org/licenses/why-not-lgpl.html

18.3 How does the BSD license affect our relationship to other projects?

The BSD license allows other projects with virtually any license, including GPL, to use our code. BSD makes it more likely that we will attract support from companies, including open-source software companies, such as Enthought and Kitware.

Any part of our code that uses (links to) GPL code, should be in a separable package.

Note that we do not have this problem with **LGPL**, which allows us to link without ourselves having a GPL.

18.4 What license does the NIH prefer?

The NIH asks that software written with NIH money can be commercialized. Quoting from: NIH NATIONAL CENTERS FOR BIOMEDICAL COMPUTING grant application document:
A software dissemination plan must be included in the application. There is no prescribed single license for software produced in this project. However NIH does have goals for software dissemination, and reviewers will be instructed to evaluate the dissemination plan relative to these goals:

1. The software should be freely available to biomedical researchers and educators in the non-profit sector, such as institutions of education, research institutes, and government laboratories.

2. The terms of software availability should permit the commercialization of enhanced or customized versions of the software, or incorporation of the software or pieces of it into other software packages.

There is more discussion of licensing in this na-mic presentation. See also these links (from the presentation):

- http://www.opensource.org

So far this might suggest that the NIH would prefer at least a BSD-like license, but the NIH has supported several GPL’d projects in imaging, AFNI being the most obvious example.
19.1 Installing graphviz on OSX

The easiest way I found to do this was using MacPorts, all other methods caused python exceptions when attempting to write out the pngs in the inheritance_diagram.py functions. Just do:

```bash
sudo port install graphviz
```

And make sure your macports directory (`/opt/local/bin`) is in your PATH.

19.2 Error writing output on OSX

If you are getting an error during the writing output... phase of the documentation build you may have a problem with your graphviz install. The error may look something like:

```bash
**writing output...** about api/generated/gen
  api/generated/nipy
  api/generated/nipy.algorithms.fwhm Format: "png" not recognized. Use one of: canon cmap cmapx cmapx_np dia dot eps fig hpgl imap imap_np ismap mif mp pcl pic plain plain-ext ps ps2 svg svgz tk vml vmlz vtx xdot
...
```

Exception occurred:

```python
File "/Users/cburns/src/nipy-repo/trunk-dev/doc/sphinxext/inheritance_diagram.py", line 238, in generate_dot
  (name, self._format_node_options(this_node_options))
IOError: [Errno 32] Broken pipe
```

Try installing graphviz using MacPorts. See the Installing graphviz on OSX for instructions.

19.3 Sphinx and reST gotchas

19.3.1 Docstrings

Sphinx and reST can be very picky about whitespace. For example, in the docstring below the Parameters section
will render correctly, where the Returns section will not. By correctly I mean Sphinx will insert a link to the CoordinateSystem class in place of the cross-reference :class:`CoordinateSystem`. The Returns section will be rendered exactly as shown below with the :class: identifier and the backticks around CoordinateSystem. This section fails because of the missing whitespace between product Coord System and the colon :

Parameters
----------
coord_systems : sequence of :class:`CoordinateSystem`

Returns
-------
product Coord System: :class:`CoordinateSystem'`
Part V

API
ALGORITHMS.CLUSTERING.BGMM

20.1 Module: algorithms.clustering.bgmm

Inheritance diagram for nipy.algorithms.clustering.bgmm:

Bayesian Gaussian Mixture Model Classes: contains the basic fields and methods of Bayesian GMMs the high level functions are/should be binded in C
The base class BGMM relies on an implementation that perfoms Gibbs sampling
A derived class VBGMM uses Variational Bayes inference instead
A third class is introduces to take advnatge of the old C-bindings, but it is limited to diagonal covariance models
Author : Bertrand Thirion, 2008-2011

20.2 Classes

20.2.1 BGMM

class nipy.algorithms.clustering.bgmm.BGMM (k=1, dim=1, means=None, precisions=None, weights=None, shrinkage=None, dof=None)

This class implements Bayesian GMMs
this class contains the folllwing fields k: int,
the number of components in the mixture

dim: int, the dimension of the data
means: array of shape (k, dim) all the means of the components
precisions: array of shape (k, dim, dim) the precisions of the components
weights: array of shape (k): weights of the mixture

shrinkage: array of shape (k): scaling factor of the posterior precisions on the mean
dof: array of shape (k) the degrees of freedom of the components

prior_means: array of shape (k, dim): the prior on the components means
prior_scale: array of shape (k, dim): the prior on the components precisions
prior_dof: array of shape (k): the prior on the dof (should be at least equal to dim)
prior_shrinkage: array of shape (k): scaling factor of the prior precisions on the mean
prior_weights: array of shape (k): the prior on the components weights

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>average_log_like(x[, tiny])</td>
<td>returns the averaged log-likelihood of the mode for the dataset x</td>
</tr>
<tr>
<td>bayes_factor(x, z[, nperm, verbose])</td>
<td>Evaluate the Bayes Factor of the current model using Chib’s method</td>
</tr>
<tr>
<td>bic(like[, tiny])</td>
<td>Computation of bic approximation of evidence</td>
</tr>
<tr>
<td>check()</td>
<td>Checking the shape of different matrices involved in the model</td>
</tr>
<tr>
<td>check_x(x)</td>
<td>essentially check that x.shape[1]==self.dim</td>
</tr>
<tr>
<td>conditional_posterior_proba(x, z[, perm])</td>
<td>Compute the probability of the current parameters of self</td>
</tr>
<tr>
<td>estimate(x[, niter, delta, verbose])</td>
<td>Estimation of the model given a dataset x</td>
</tr>
<tr>
<td>evidence(x, z[, nperm, verbose])</td>
<td>See bayes_factor(self, x, z, nperm=0, verbose=0)</td>
</tr>
<tr>
<td>guess_priors(x[, nocheck])</td>
<td>Set the priors in order of having them weakly uninformative</td>
</tr>
<tr>
<td>guess_regularizing(x[, bcheck])</td>
<td>Set the regularizing priors as weakly informative</td>
</tr>
<tr>
<td>initialize(x)</td>
<td>initialize z using a k-means algorithm, then update the parameters</td>
</tr>
<tr>
<td>initialize_and_estimate(x[, z, niter, ...])</td>
<td>Estimation of self given x</td>
</tr>
<tr>
<td>likelihood(x)</td>
<td>return the likelihood of the model for the data x</td>
</tr>
<tr>
<td>map_label(x[, like])</td>
<td>return the MAP labelling of x</td>
</tr>
<tr>
<td>mixture_likelihood(x)</td>
<td>Returns the likelihood of the mixture for x</td>
</tr>
<tr>
<td>plugin(means, precisions, weights)</td>
<td>Set manually the weights, means and precision of the model</td>
</tr>
<tr>
<td>pop(z)</td>
<td>compute the population, i.e. the statistics of allocation</td>
</tr>
<tr>
<td>probability_under_prior()</td>
<td>Compute the probability of the current parameters of self</td>
</tr>
<tr>
<td>sample(x[, niter, mem, verbose])</td>
<td>sample the indicator and parameters</td>
</tr>
<tr>
<td>sample_and_average(x[, niter, verbose])</td>
<td>sample the indicator from the likelihood</td>
</tr>
<tr>
<td>sample_indicator(like)</td>
<td></td>
</tr>
<tr>
<td>set_priors(prior_means, prior_weights, ...)</td>
<td>Set the prior of the BGMM</td>
</tr>
<tr>
<td>show(x, gd[, density, axes])</td>
<td>Function to plot a GMM, still in progress</td>
</tr>
<tr>
<td>show_components(x, gd[, density, mpaxes])</td>
<td>Function to plot a GMM – Currently, works only in 1D</td>
</tr>
<tr>
<td>test(x[, tiny])</td>
<td>Returns the log-likelihood of the mixture for x</td>
</tr>
<tr>
<td>train(x[, z, niter, delta, ninit, verbose])</td>
<td>Idem initialize_and_estimate</td>
</tr>
<tr>
<td>unweighted_likelihood(x)</td>
<td>return the likelihood of each data for each component</td>
</tr>
<tr>
<td>unweighted_likelihood_(x)</td>
<td>return the likelihood of each data for each component</td>
</tr>
<tr>
<td>update(x, z)</td>
<td>update function (draw a sample of the GMM parameters)</td>
</tr>
<tr>
<td>update_means(x, z)</td>
<td>Given the allocation vector z,</td>
</tr>
<tr>
<td>update_precisions(x, z)</td>
<td>Given the allocation vector z,</td>
</tr>
<tr>
<td>update_weights(z)</td>
<td>Given the allocation vector z, resample the weights parameter</td>
</tr>
</tbody>
</table>
__init__ (k=1, dim=1, means=None, precisions=None, weights=None, shrinkage=None, dof=None)
Initialize the structure with the dimensions of the problem Eventually provide different terms

average_log_like (x, tiny=1e-15)
returns the averaged log-likelihood of the mode for the dataset x

Parameters x: array of shape (n_samples,self.dim):
the data used in the estimation process
tiny = 1.e-15: a small constant to avoid numerical singularities:

bayes_factor (x, z, nperm=0, verbose=0)
Evaluate the Bayes Factor of the current model using Chib’s method

Parameters x: array of shape (nb_samples,dim):
the data from which bic is computed
z: array of shape (nb_samples), type = np.int:
the corresponding classification
nperm=0: int:
the number of permutations to sample to model the label switching issue in the computation of the Bayes Factor By default, exhaustive permutations are used
verbose=0: verbosity mode:

Returns bf (float) the computed evidence (Bayes factor):

Notes

bic (like, tiny=1e-15)
Computation of bic approximation of evidence

Parameters like, array of shape (n_samples, self.k):
component-wise likelihood
tiny=1.e-15, a small constant to avoid numerical singularities:

Returns the bic value, float:

check ()
Checking the shape of different matrices involved in the model
check_x (x)

esentially check that x.shape[1]==self.dim
x is returned with possibly reshaping

conditional_posterior_proba (x, z, perm=None)
Compute the probability of the current parameters of self given x and z

Parameters x: array of shape (nb_samples, dim),:
the data from which bic is computed
z: array of shape (nb_samples), type = np.int,:
the corresponding classification
perm: array ok shape(nperm, self.k), typ=np.int, optional:
    all permutation of z under which things will be recomputed. By default, no permutation is performed.

estimate (x, niter=100, delta=0.0001, verbose=0)
    Estimation of the model given a dataset x
    
    Parameters
    x array of shape (n_samples, dim):
        the data from which the model is estimated
    niter=100: maximal number of iterations in the estimation process:
    delta = 1.e-4: increment of data likelihood at which convergence is declared
    verbose=0: verbosity mode:
    
    Returns
    bic: an asymptotic approximation of model evidence

evidence (x, z, nperm=0, verbose=0)
    See bayes_factor(self, x, z, nperm=0, verbose=0)

guess_priors (x, nocheck=0)
    Set the priors in order of having them weakly uninformative. This is from Fraley and Raftery; Journal of Classification 24:155-181 (2007)
    
    Parameters
    x, array of shape (nb_samples, self.dim):
        the data used in the estimation process
    nocheck: boolean, optional:
        if nocheck==True, check is skipped

guess_regularizing (x, bcheck=1)
    Set the regularizing priors as weakly informative according to Fraley and Raftery; Journal of Classification 24:155-181 (2007)
    
    Parameters
    x, array of shape (n_samples, dim):
        the data used in the estimation process

initialize (x)
    initialize z using a k-means algorithm, then update the parameters
    
    Parameters
    x: array of shape (nb_samples, self.dim):
        the data used in the estimation process

initialize_and_estimate (x, z= None, niter=100, delta=0.0001, ninit=1, verbose=0)
    Estimation of self given x
    
    Parameters
    x array of shape (n_samples, dim):
        the data from which the model is estimated
    z = None: array of shape (n_samples):
        a prior labelling of the data to initialize the computation
    niter=100: maximal number of iterations in the estimation process:
    delta = 1.e-4: increment of data likelihood at which convergence is declared
ninit=1: number of initialization performed:

to reach a good solution

verbose=0: verbosity mode:

Returns the best model is returned:

\textbf{likelihood}(x)

\text{return the likelihood of the model for the data x the values are weighted by the components weights}

Parameters \text{x array of shape (n_samples, self.dim)}:

the data used in the estimation process

Returns \text{like, array of shape(n_samples, self.k)}:

component-wise likelihood

\textbf{map\_label}(x, like=None)

\text{return the MAP labelling of x}

Parameters \text{x array of shape (n_samples, dim)}:

the data under study

like=None array of shape (n_samples, self.k):

component-wise likelihood if like==None, it is recomputed

Returns \text{z: array of shape(n_samples): the resulting MAP labelling}:

of the rows of x

\textbf{mixture\_likelihood}(x)

\text{Returns the likelihood of the mixture for x}

Parameters \text{x: array of shape (n_samples, self.dim)}:

the data used in the estimation process

\textbf{plugin}(means, precisions, weights)

Set manually the weights, means and precision of the model

Parameters \text{means: array of shape (self.k, self.dim)}:

\text{precisions: array of shape (self.k, self.dim, self.dim)}:

\text{or (self.k, self.dim)}

\text{weights: array of shape (self.k)}:

\textbf{pop}(z)

\text{compute the population, i.e. the statistics of allocation}

Parameters \text{z array of shape (nb_samples), type = np.int}:

the allocation variable

Returns \text{hist : array shape (self.k) count variable}

\textbf{probability\_under\_prior}()

\text{Compute the probability of the current parameters of self given the priors}

\textbf{sample}(x, niter=1, mem=0, verbose=0)

\text{sample the indicator and parameters}

Parameters \text{x array of shape (nb_samples, self.dim)}:
the data used in the estimation process

**niter=1**: the number of iterations to perform

**mem=0**: if mem, the best values of the parameters are computed

**verbose=0**: verbosity mode

**Returns**

- **best_weights**: array of shape (self.k):
- **best_means**: array of shape (self.k, self.dim):
- **best_precisions**: array of shape (self.k, self.dim, self.dim):
- **possibleZ**: array of shape (nb_samples, niter):

the z that give the highest posterior to the data is returned first

**sample_and_average** *(x, niter=1, verbose=0)*

sample the indicator and parameters the average values for weights, means, precisions are returned

**Parameters**

- **x**: array of shape (nb_samples, dim):
  
  the data from which bic is computed
- **niter=1**: number of iterations:

**Returns**

- **weights**: array of shape (self.k):
- **means**: array of shape (self.k, self.dim):
- **precisions**: array of shape (self.k, self.dim, self.dim):

  or (self.k, self.dim) these are the average parameters across samplings

**Notes**

All this makes sense only if no label switching as occurred so this is wrong in general (asymptotically).

fix: implement a permutation procedure for components identification

**sample_indicator** *(like)*

sample the indicator from the likelihood

**Parameters**

- **like**: array of shape (nb_samples, self.k):
  
  component-wise likelihood

**Returns**

- **z**: array of shape (nb_samples): a draw of the membership variable:

**set_priors** *(prior_means, prior_weights, prior_scale, prior_dof, prior_shrinkage)*

Set the prior of the BGMM

**Parameters**

- **prior_means**: array of shape (self.k, self.dim):
- **prior_weights**: array of shape (self.k):
- **prior_scale**: array of shape (self.k, self.dim, self.dim):
- **prior_dof**: array of shape (self.k):
- **prior_shrinkage**: array of shape (self.k):

**show** *(x, gd, density=None, axes=None)*

Function to plot a GMM, still in progress Currently, works only in 1D and 2D

**Parameters**

- **x**: array of shape (n_samples, dim):
the data under study

gd: GridDescriptor instance:

density: array os shape(prod(gd.n_bins)):

density of the model one the discrete grid implied by gd by default, this is recomputed

show_components(x, gd, density=None, mpaxes=None)
Function to plot a GMM – Currently, works only in 1D

Parameters x: array of shape(n_samples, dim):

the data under study

gd: GridDescriptor instance:

density: array os shape(prod(gd.n_bins)):

density of the model one the discrete grid implied by gd by default, this is recomputed

mpaxes: axes handle to make the figure, optional:

if None, a new figure is created

test(x, tiny=1e-15)
Returns the log-likelihood of the mixture for x

Parameters x array of shape (n_samples, self.dim):

the data used in the estimation process

Returns ll: array of shape(n_samples):

the log-likelihood of the rows of x

train(x, z=None, niter=100, delta=0.0001, ninit=1, verbose=0)
Idem initialize_and_estimate

unweighted_likelihood(x)
return the likelihood of each data for each component the values are not weighted by the component weights

Parameters x: array of shape (n_samples, self.dim):

the data used in the estimation process

Returns like, array of shape(n_samples, self.k):

unweighted component-wise likelihood

Notes

Hopefully faster

unweighted_likelihood_(x)
return the likelihood of each data for each component the values are not weighted by the component weights

Parameters x: array of shape (n_samples, self.dim):

the data used in the estimation process

Returns like, array of shape(n_samples, self.k):

unweighted component-wise likelihood
update \((x, z)\)
update function (draw a sample of the GMM parameters)

**Parameters**
- \(x\) array of shape \((\text{nb}\_\text{samples}, \text{self}.\text{dim})\):
  - the data used in the estimation process
- \(z\) array of shape \((\text{nb}\_\text{samples})\), type = np.int:
  - the corresponding classification

update_means \((x, z)\)
Given the allocation vector \(z\), and the corresponding data \(x\), resample the mean

**Parameters**
- \(x\): array of shape \((\text{nb}\_\text{samples}, \text{self}.\text{dim})\):
  - the data used in the estimation process
- \(z\): array of shape \((\text{nb}\_\text{samples})\), type = np.int:
  - the corresponding classification

update_precisions \((x, z)\)
Given the allocation vector \(z\), and the corresponding data \(x\), resample the precisions

**Parameters**
- \(x\) array of shape \((\text{nb}\_\text{samples}, \text{self}.\text{dim})\):
  - the data used in the estimation process
- \(z\) array of shape \((\text{nb}\_\text{samples})\), type = np.int:
  - the corresponding classification

update_weights \((z)\)
Given the allocation vector \(z\), resample the weights parameter

**Parameters**
- \(z\) array of shape \((\text{nb}\_\text{samples})\), type = np.int:
  - the allocation variable

20.2.2 VBGMM

class nipy.algorithms.clustering.bgmm.VBGMM \((k=1, \text{dim}=1, \text{means}={\text{None}}, \text{precisions}={\text{None}}, \text{weights}={\text{None}}, \text{shrinkage}={\text{None}}, \text{dof}={\text{None}})\)

**Bases:** nipy.algorithms.clustering.bgmm.BGMM

Subclass of Bayesian GMMs (BGMM) that implements Variational Bayes estimation of the parameters

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>average_log_like(x[, tiny])</td>
<td>returns the averaged log-likelihood of the mode for the dataset x</td>
</tr>
<tr>
<td>bayes_factor(x, z[, nperm, verbose])</td>
<td>Evaluate the Bayes Factor of the current model using Chib's method</td>
</tr>
<tr>
<td>bic(like[, tiny])</td>
<td>Computation of bic approximation of evidence</td>
</tr>
<tr>
<td>check()</td>
<td>Checking the shape of different matrices involved in the model</td>
</tr>
<tr>
<td>check_x(x)</td>
<td>essentially check that x.shape[1]==self.dim</td>
</tr>
<tr>
<td>conditional_posterior_proba(x, z[, perm])</td>
<td>Compute the probability of the current parameters of self</td>
</tr>
<tr>
<td>estimate(x[, niter, delta, verbose])</td>
<td>estimation of self given x</td>
</tr>
<tr>
<td>evidence(x[, like, verbose])</td>
<td>computation of evidence bound aka free energy</td>
</tr>
<tr>
<td>guess_priors(x[, nocheck])</td>
<td>Set the priors in order of having them weakly uninformative</td>
</tr>
</tbody>
</table>

Continued on next page
Table 20.2 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>guess_regularizing(x[, bcheck])</code></td>
<td>Set the regularizing priors as weakly informative</td>
</tr>
<tr>
<td><code>initialize(x)</code></td>
<td>Initialize z using a k-means algorithm, then update the parameters</td>
</tr>
<tr>
<td><code>initialize_and_estimate(x[, z, niter, ...])</code></td>
<td>Estimation of self given x</td>
</tr>
<tr>
<td><code>likelihood(x)</code></td>
<td>Return the likelihood of the model for the data x</td>
</tr>
<tr>
<td><code>map_label(x[, like])</code></td>
<td>Return the MAP labelling of x</td>
</tr>
<tr>
<td><code>mixture_likelihood(x)</code></td>
<td>Returns the likelihood of the mixture for x</td>
</tr>
<tr>
<td><code>plugin(means, precisions, weights)</code></td>
<td>Set manually the weights, means and precision of the model</td>
</tr>
<tr>
<td><code>pop(like[, tiny])</code></td>
<td>Compute the population, i.e. the statistics of allocation</td>
</tr>
<tr>
<td><code>probability_under_prior()</code></td>
<td>Compute the probability of the current parameters of self</td>
</tr>
<tr>
<td><code>sample(x[, niter, mem, verbose])</code></td>
<td>Sample the indicator and parameters</td>
</tr>
<tr>
<td><code>sample_and_average(x[, niter, verbose])</code></td>
<td>Sample the indicator from the likelihood</td>
</tr>
<tr>
<td><code>sample_indicator(like)</code></td>
<td>Sample the indicator from the likelihood</td>
</tr>
<tr>
<td><code>set_priors(prior_means, prior_weights, ...)</code></td>
<td>Set the prior of the BGMM</td>
</tr>
<tr>
<td><code>show(x, gd[, density, axes])</code></td>
<td>Function to plot a GMM, still in progress</td>
</tr>
<tr>
<td><code>show_components(x, gd[, density, mpaxes])</code></td>
<td>Function to plot a GMM – Currently, works only in 1D</td>
</tr>
<tr>
<td><code>test(x[, tiny])</code></td>
<td>Returns the log-likelihood of the mixture for x</td>
</tr>
<tr>
<td><code>train(x[, z, niter, delta, ninit, verbose])</code></td>
<td>Idem initialize_and_estimate</td>
</tr>
<tr>
<td><code>unweighted_likelihood(x)</code></td>
<td>Return the likelihood of each data for each component</td>
</tr>
<tr>
<td><code>unweighted_likelihood(x)</code></td>
<td>Return the likelihood of all data for each component</td>
</tr>
<tr>
<td><code>update(x, z)</code></td>
<td>Update function (draw a sample of the GMM parameters)</td>
</tr>
<tr>
<td><code>update_means(x, z)</code></td>
<td>Given the allocation vector z,</td>
</tr>
<tr>
<td><code>update_precisions(x, z)</code></td>
<td>Given the allocation vector z,</td>
</tr>
<tr>
<td><code>update_weights(z)</code></td>
<td>Given the allocation vector z, resample the weights parameter</td>
</tr>
</tbody>
</table>

---

**__init__**(k=1, dim=1, means=None, precisions=None, weights=None, shrinkage=None, dof=None)

**average_log_like**(x, tiny=1e-15)

returns the averaged log-likelihood of the mode for the dataset x

**Parameters**

- **x**: array of shape (n_samples, self.dim):
  - the data used in the estimation process
- **tiny** = 1.e-15: a small constant to avoid numerical singularities:

**bayes_factor**(x, z, nperm=0, verbose=0)

Evaluate the Bayes Factor of the current model using Chib’s method

**Parameters**

- **x**: array of shape (nb_samples, dim):
  - the data from which bic is computed
- **z**: array of shape (nb_samples), type = np.int:
  - the corresponding classification
- **nperm**: int:
  - the number of permutations to sample to model the label switching issue in the computation of the Bayes Factor By default, exhaustive permutations are used
- **verbose**: verbosity mode:

**Returns**

- **bf** (float) the computed evidence (Bayes factor):

---

20.2. Classes 157

---
Notes


**bic(like, tiny=1e-15)**

Computation of bic approximation of evidence

**Parameters**
- `like`, array of shape (n_samples, self.k): component-wise likelihood
- `tiny=1.e-15`, a small constant to avoid numerical singularities

**Returns**
- the bic value, float

**check()**

Checking the shape of different matrices involved in the model

**check_x(x)**

essentially check that x.shape[1]==self.dim

x is returned with possibly reshaping

**conditional_posterior_proba(x, z, perm=None)**

Compute the probability of the current parameters of self given x and z

**Parameters**
- `x`: array of shape (nb_samples, dim), the data from which bic is computed
- `z`: array of shape (nb_samples), type = np.int, the corresponding classification
- `perm`: array of shape(nperm, self.k), typ=np.int, optional: all permutation of z under which things will be recomputed By default, no permutation is performed

**estimate(x, niter=100, delta=0.0001, verbose=0)**

estimation of self given x

**Parameters**
- `x` array of shape (nb_samples,dim): the data from which the model is estimated
- `z = None`: array of shape (nb_samples): a prior labelling of the data to initialize the computation
- `niter=100`: maximal number of iterations in the estimation process
- `delta = 1.e-4`: increment of data likelihood at which convergence is declared
- `verbose=0`: verbosity mode

**evidence(x, like=None, verbose=0)**

computation of evidence bound aka free energy

**Parameters**
- `x` array of shape (nb_samples,dim): the data from which evidence is computed
like=None: array of shape (nb_samples, self.k), optional :
    component-wise likelihood If None, it is recomputed

verbose=0: verbosity model :

Returns ev (float) the computed evidence :

guess_priors (x, nocheck=0)
    Set the priors in order of having them weakly uninformative this is from Fraley and raftery; Journal of Classification 24:155-181 (2007)

Parameters x, array of shape (nb_samples,self.dim) :
    the data used in the estimation process

nocheck: boolean, optional, :
    if nocheck==True, check is skipped

guess_regularizing (x, bcheck=1)
    Set the regularizing priors as weakly informative according to Fraley and raftery; Journal of Classification 24:155-181 (2007)

Parameters x array of shape (n_samples,dim) :
    the data used in the estimation process

initialize (x)
    initialize z using a k-means algorithm, then upate the parameters

Parameters x: array of shape (nb_samples,self.dim) :
    the data used in the estimation process

initialize_and_estimate (x, z=None, niter=100, delta=0.0001, ninit=1, verbose=0)
    Estimation of self given x

Parameters x array of shape (n_samples,dim) :
    the data from which the model is estimated

z = None: array of shape (n_samples) :
    a prior labelling of the data to initialize the computation

niter=100: maximal number of iterations in the estimation process :

delta = 1.e-4: increment of data likelihood at which :
    convergence is declared

ninit=1: number of initialization performed :
    to reach a good solution

verbose=0: verbosity mode :

Returns the best model is returned :

likelihood (x)
    return the likelihood of the model for the data x the values are weighted by the components weights

Parameters x: array of shape (nb_samples, self.dim) :
    the data used in the estimation process

Returns like: array of shape(nb_samples, self.k) :
component-wise likelihood

map_label(x, like=None)
return the MAP labelling of x

Parameters x array of shape (nb_samples, dim):
the data under study

like=None array of shape (nb_samples, self.k):
component-wise likelihood if like==None, it is recomputed

Returns z: array of shape (nb_samples):
the resulting MAP labelling:
of the rows of x

mixture_likelihood(x)
Returns the likelihood of the mixture for x

Parameters x: array of shape (n_samples, self.dim):
the data used in the estimation process

plugin(means, precisions, weights)
Set manually the weights, means and precision of the model

Parameters means: array of shape (self.k, self.dim):
precisions: array of shape (self.k, self.dim, self.dim):
or (self.k, self.dim)
weights: array of shape (self.k):

pop(like, tiny=1e-15)
compute the population, i.e. the statistics of allocation

Parameters like array of shape (nb_samples, self.k):
the likelihood of each item being in each class

probability_under_prior()
Compute the probability of the current parameters of self given the priors

sample(x, niter=1, mem=0, verbose=0)
sample the indicator and parameters

Parameters x array of shape (nb_samples, self.dim):
the data used in the estimation process

niter=1: the number of iterations to perform

mem=0: if mem, the best values of the parameters are computed:

verbose=0: verbosity mode:

Returns best_weights: array of shape (self.k):
best_means: array of shape (self.k, self.dim):
best_precisions: array of shape (self.k, self.dim, self.dim):
possibleZ: array of shape (nb_samples, niter):
the z that give the highest posterior to the data is returned first
**sample_and_average** *(x, niter=1, verbose=0)*

Sample the indicator and parameters the average values for weights, means, precisions are returned.

**Parameters**  
x = array of shape (nb_samples, dim):  
the data from which bic is computed

niter=1: number of iterations:

**Returns**  
weights: array of shape (self.k):

means: array of shape (self.k, self.dim):

precisions: array of shape (self.k, self.dim, self.dim):

or (self.k, self.dim) these are the average parameters across samplings

**Notes**

All this makes sense only if no label switching as occurred so this is wrong in general (asymptotically).

fix: implement a permutation procedure for components identification

**sample_indicator** *(like)*

Sample the indicator from the likelihood

**Parameters**  
like: array of shape (nb_samples, self.k):

cOMPONENT-WISE likelihood

**Returns**  
z: array of shape (nb_samples): a draw of the membership variable:

**set_priors** *(prior_means, prior_weights, prior_scale, prior_dof, prior_shrinkage)*

Set the prior of the BGMM

**Parameters**  
prior_means: array of shape (self.k, self.dim):

prior_weights: array of shape (self.k):

prior_scale: array of shape (self.k, self.dim, self.dim):

prior_dof: array of shape (self.k):

prior_shrinkage: array of shape (self.k):

**show** *(x, gd, density=None, axes=None)*

Function to plot a GMM, still in progress Currently, works only in 1D and 2D

**Parameters**  
x: array of shape (n_samples, dim):

the data under study

gd: GridDescriptor instance:

density: array os shape(prod(gd.n_bins)):

density of the model one the discrete grid implied by gd by default, this is recomputed

**show_components** *(x, gd, density=None, mpaxes=None)*

Function to plot a GMM – Currently, works only in 1D

**Parameters**  
x: array of shape (n_samples, dim):

the data under study
gd: GridDescriptor instance

density: array os shape(prod(gd.n_bins)):

density of the model one the discrete grid implied by gd by default, this is recomputed

mpaxes: axes handle to make the figure, optional, :

if None, a new figure is created

test (x, tiny=1e-15)

Returns the log-likelihood of the mixture for x

Parameters  x array of shape (n_samples,self.dim) :

the data used in the estimation process

Returns  ll: array of shape(n_samples) :

the log-likelihood of the rows of x

train (x, z=None, niter=100, delta=0.0001, ninit=1, verbose=0)

Idem initialize_and_estimate

unweighted likelihood (x)

return the likelihood of each data for each component the values are not weighted by the component weights

Parameters  x: array of shape (n_samples,self.dim) :

the data used in the estimation process

Returns  like, array of shape(n_samples,self.k) :

unweighted component-wise likelihood

Notes

Hopefully faster

unweighted likelihood (x)

return the likelihood of each data for each component the values are not weighted by the component weights

Parameters  x: array of shape (n_samples,self.dim) :

the data used in the estimation process

Returns  like, array of shape(n_samples,self.k) :

unweighted component-wise likelihood

update (x, z)

update function (draw a sample of the GMM parameters)

Parameters  x array of shape (nb_samples,self.dim) :

the data used in the estimation process

z array of shape (nb_samples), type = np.int :

the corresponding classification

update_means (x, z)

Given the allocation vector z, and the corresponding data x, resample the mean
Parameters $x$: array of shape (nb_samples, self.dim):

the data used in the estimation process

$z$: array of shape (nb_samples), type = np.int:

the corresponding classification

$\text{update_precisions}(x, z)$

Given the allocation vector $z$, and the corresponding data $x$, resample the precisions

Parameters $x$ array of shape (nb_samples, self.dim):

the data used in the estimation process

$z$ array of shape (nb_samples), type = np.int:

the corresponding classification

$\text{update_weights}(z)$

Given the allocation vector $z$, resample the weights parameter

Parameters $z$ array of shape (nb_samples), type = np.int:

the allocation variable

## 20.3 Functions

nipy.algorithms.clustering.bgmm.$\text{detsh}(H)$

Routine for the computation of determinants of symmetric positive matrices

Parameters $H$ array of shape(n,n):

the input matrix, assumed symmetric and positive

Returns $dh$: float, the determinant:

nipy.algorithms.clustering.bgmm.$\text{dirichlet_eval}(w, alpha)$

Evaluate the probability of a certain discrete draw $w$ from the Dirichlet density with parameters $alpha$

Parameters $w$: array of shape (n):

$alpha$: array of shape (n):

nipy.algorithms.clustering.bgmm.$\text{dkl_dirichlet}(w1, w2)$

Returns the KL divergence between two dirichlet distribution

Parameters $w1$: array of shape(n),:

the parameters of the first dirichlet density

$w2$: array of shape(n),:

the parameters of the second dirichlet density

nipy.algorithms.clustering.bgmm.$\text{dkl_gaussian}(m1, P1, m2, P2)$

Returns the KL divergence between gausians densities

Parameters $m1$: array of shape (n),:

the mean parameter of the first density

$P1$: array of shape(n,n),:

the precision parameters of the first density
m2: array of shape (n), :
    the mean parameter of the second density

P2: array of shape(n,n), :
    the precision parameters of the second density

nipy.algorithms.clustering.bgmm.dkl_wishart(a1, B1, a2, B2)
    returns the KL divergence between two Wishart distribution of parameters (a1,B1) and (a2,B2).

Parameters  a1: Float, :
    degrees of freedom of the first density

B1: array of shape(n,n), :
    scale matrix of the first density

a2: Float, :
    degrees of freedom of the second density

B2: array of shape(n,n), :
    scale matrix of the second density

Returns  dkl: float, the Kullback-Leibler divergence :

nipy.algorithms.clustering.bgmm.generate_Wishart(n, V)
    Generate a sample from Wishart density

Parameters  n: float, :
    the number of degrees of freedom of the Wishart density

V: array of shape (n,n) :
    the scale matrix of the Wishart density

Returns  W: array of shape (n,n) :
    the draw from Wishart density

nipy.algorithms.clustering.bgmm.generate_normals(m, P)
    Generate a Gaussian sample with mean m and precision P

Parameters  m array of shape n: the mean vector :

P array of shape (n,n): the precision matrix :

Returns  ng : array of shape(n): a draw from the gaussian density

nipy.algorithms.clustering.bgmm.generate_perm(k, nperm=100)
    returns an array of shape(nperm,k) representing the permutations of k elements

Parameters  k, int the number of elements to be permuted :

nperm=100 the maximal number of permutations :
    if gamma(k+1)>nperm: only nperm random draws are generated :

Returns  p: array of shape(nperm,k): each row is permutation of k :

nipy.algorithms.clustering.bgmm.multinomial(probabilities)
    Generate samples form a multivariate distribution

Parameters  probabilities: array of shape (nelements, nclasses): :
likelihood of each element belonging to each class each row is assumed to sum to 1. One sample is drawn from each row, resulting in

Returns \( z \) array of shape (nelements): the draws, that take values in \([0..\text{nclasses}-1]\)

\[
\text{nipy.algorithms.clustering.bgmm.normal_eval}(\mu, P, x, dP=None)
\]

Probability of \( x \) under \( \text{normal}(\mu, \text{inv}(P)) \)

Parameters
\( \mu \): array of shape (n), the mean parameter
\( P \): array of shape (n, n), the precision matrix
\( x \): array of shape (n), the data to be evaluated

Returns \((\text{float})\) the density:

\[
\text{nipy.algorithms.clustering.bgmm.wishart_eval}(n, V, W, dV=None, dW=None, piV=None)
\]

Evaluation of the probability of \( W \) under \( \text{Wishart}(n,V) \)

Parameters
\( n \): float, the number of degrees of freedom (dofs)
\( V \): array of shape (n,n), the scale matrix of the Wishart density
\( W \): array of shape (n,n), the sample to be evaluated
\( dV \): float, optional, determinant of \( V \)
\( dW \): float, optional, determinant of \( W \)
\( piV \): array of shape (n,n), optional, inverse of \( V \)

Returns \((\text{float})\) the density:
ALGORITHMS.CLUSTERING.GGMIXTURE

21.1 Module: algorithms.clustering.ggmixture

Inheritance diagram for nipy.algorithms.clustering.ggmixture:

- clustering.ggmixture.GGGM
- clustering.ggmixture.GGM
- clustering.ggmixture.Gamma

One-dimensional Gamma-Gaussian mixture density classes: Given a set of points the algo provides approximate maximum likelihood estimates of the mixture distribution using an EM algorithm.

Author: Bertrand Thirion and Merlin Keller 2005-2008

21.2 Classes

21.2.1 GGGM

class nipy.algorithms.clustering.ggmixture.GGGM(shape_n=1, scale_n=1, mean=0, var=1, shape_p=1, scale_p=1, mixt=array([0.33333333, 0.33333333, 0.33333333]))

Bases: object

The basic one dimensional Gamma-Gaussian-Gamma Mixture estimation class, where the first gamma has a negative sign, while the second one has a positive sign.

7 parameters are used: - shape_n: negative gamma shape - scale_n: negative gamma scale - mean: gaussian mean - var: gaussian variance - shape_p: positive gamma shape - scale_p: positive gamma scale - mixt: array of mixture parameter (weights of the n-gamma, gaussian and p-gamma)
Methods

**Estep**(x)  
Update probabilistic memberships of the three components

**Mstep**(x, z)  
Mstep of the estimation:

**component_likelihood**(x)  
Compute the likelihood of the data x under

**estimate**(x[, niter, delta, bias, verbose, ...])  
Whole EM estimation procedure:

**init**(x[, mixt])  
initialization of the different parameters

**init_fdr**(x[, dof, copy])  
Initilization of the class based on a fdr heuristic: the probability to be in the positive component

**parameters**()  
Print the parameters

**posterior**(x)  
Compute the posterior probability of the three components

**show**(x[, mpaxes])  
Visualization of mixture shown on the empirical histogram of x

---

```python
__init__ (shape_n=1, scale_n=1, mean=0, var=1, shape_p=1, scale_p=1, mixt=array([ 0.33333333, 0.33333333, 0.33333333]))
```

Constructor

**Parameters**

- **shape_n**: float, optional  
parameters of the negative gamma; must be positive
- **scale_n**: float, optional  
parameters of the negative gamma; must be positive
- **mean**: float, optional  
parameters of the gaussian; must be positive
- **var**: float, optional  
parameters of the gaussian; must be positive
- **shape_p**: float, optional  
parameters of the positive gamma; must be positive
- **scale_p**: float, optional  
parameters of the positive gamma; must be positive
- **mixt**: array of shape (3,), optional  
the mixing proportions; they should be positive and sum to 1

**Estep**(x)  
Update probabilistic memberships of the three components

**Parameters**

- **x**: array of shape (nbitems,)  
the input data

**Returns**

- **z**: ndarray of shape (nbitems, 3)  
probabilistic membership

**Notes**

z[0, :] is the membership the negative gamma z[1, :] is the membership of the gaussian z[2, :] is the membership of the positive gamma

**Mstep**(x, z)  
Mstep of the estimation: Maximum likelihood update the parameters of the three components

**Parameters**

- **x**: array of shape (nbitem,)  
input data
z: array of shape (nbitems,3):
  probabilistic membership

component_likelihood(x)
  Compute the likelihood of the data x under the three components negative gamma, gaussina, positive
  gaussian

  Parameters  x: array of shape (nbitem,):
               the data under evaluation

  Returns   ng, y, pg: three arrays of shape(nbitem):
               The likelihood of the data under the 3 components

estimate(x, niter=100, delta=0.0001, bias=0, verbose=0, gaussian_mix=0)
  Whole EM estimation procedure:

  Parameters  x: array of shape (nbitem):
               input data

    niter: integer, optional:
      max number of iterations

    delta: float, optional:
      increment in LL at which convergence is declared

    bias: float, optional:
      lower bound on the gaussian variance (to avoid shrinkage)

    gaussian_mix: float, optional:
      if nonzero, lower bound on the gaussian mixing weight (to avoid shrinkage)

    verbose: 0, 1 or 2:
      verbosity level

  Returns  z: array of shape (nbitem, 3):
               the membership matrix

init(x, mixt=None)
  initialization of the different parameters

  Parameters  x: array of shape(nbitems):
               the data to be processed

    mixt: None or array of shape(3), optional
      prior mixing proportions. If None, the classes have equal weight

init_fdr(x, dof=-1, copy=True)
  Initializiation of the class based on a fdr heuristic: the probability to be in the positive component is propor-
  tional to the ‘positive fdr’ of the data. The same holds for the negative part. The point is that the gamma
  parts should model nothing more that the tails of the distribution.

  Parameters  x: array of shape(nbitem):
               the data under consideration

    dof: integer, optional:
number of degrees of freedom if x is thought to be a student variate. By default, it is
handled as a normal

**copy**: boolean, optional :

If True, copy the data.

```
parameters ()
```

Print the parameters

```
posterior (x)
```

Compute the posterior probability of the three components given the data

**Parameters**  
```
x: array of shape (nbitem,)
```

the data under evaluation

**Returns**  
```
ng, y, pg: three arrays of shape(nbitem) :
```

the posteriori of the 3 components given the data

**Notes**

ng + y + pg = np.ones(nbitem)

```
show (x, mpaxes=None)
```

Visualization of mixture shown on the empirical histogram of x

**Parameters**  
```
x: ndarray of shape (nditem,):
```

data

**mpaxes**: matplotlib axes, optional :

axes handle used for the plot if None, new axes are created.

### 21.2.2 GGM

```
class nipy.algorithms.clustering.ggmixture.GGM (shape=1, scale=1, mean=0, var=1, mixt=0.5)
```

Bases: object

This is the basic one dimensional Gaussian-Gamma Mixture estimation class Note that it can work with positive
or negative values, as long as there is at least one positive value. NB : The gamma distribution is defined only
on positive values.

5 scalar members - mean: gaussian mean - var: gaussian variance (non-negative) - shape: gamma shape (non-
negative) - scale: gamma scale (non-negative) - mixt: mixture parameter (non-negative, weight of the gamma)

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Estep(x)</td>
<td>E step of the estimation:</td>
</tr>
<tr>
<td>Mstep(x, z)</td>
<td>Mstep of the model: maximum likelihood</td>
</tr>
<tr>
<td>estimate(x[, niter, delta, verbose])</td>
<td>Complete EM estimation procedure</td>
</tr>
<tr>
<td>parameters()</td>
<td>print the parameters of self</td>
</tr>
<tr>
<td>posterior(x)</td>
<td>Posterior probability of observing the data x for each component</td>
</tr>
<tr>
<td>show(x)</td>
<td>Visualization of the mm based on the empirical histogram of x</td>
</tr>
</tbody>
</table>
__init__ (shape=1, scale=1, mean=0, var=1, mixt=0.5)

Estep (x)
E step of the estimation: Estimation of ata membership

Parameters  x: array of shape (nbitems,):
input data

Returns  z: array of shape (nbitems, 2):
the membership matrix

Mstep (x, z)
Mstep of the model: maximum likelihood estimation of the parameters of the model

Parameters  x: array of shape (nbitems,)
input data

z array of shape (nbitems, 2):
the membership matrix

estimate (x, niter=10, delta=0.0001, verbose=False)
Complete EM estimation procedure

Parameters  x: array of shape (nbitems,)
the data to be processed

niter : int, optional
max nb of iterations

delta : float, optional
criterion for convergence

verbose : bool, optional
If True, print values during iterations

Returns  LL, float:
average final log-likelihood

parameters ()
print the parameters of self

posterior (x)
Posterior probability of observing the data x for each component

Parameters  x: array of shape (nbitems,):
the data to be processed

Returns  y, pg: arrays of shape (nbitem)
the posterior probability

show (x)
Visualization of the mm based on the empirical histogram of x

Parameters  x: array of shape (nbitems,)
the data to be processed
21.2.3 Gamma

class nipy.algorithms.clustering.ggmixture.Gamma(shape=1, scale=1)
    Bases: object

    Basic one dimensional Gaussian-Gamma Mixture estimation class

    Note that it can work with positive or negative values, as long as there is at least one positive value. NB : The
gamma distribution is defined only on positive values. 5 parameters are used: - mean: gaussian mean - var:
gaussian variance - shape: gamma shape - scale: gamma scale - mixt: mixture parameter (weight of the gamma)

Methods

    __init__(shape=1, scale=1)
    check(x)
    estimate(x[, eps])  # ML estimation of the Gamma parameters
    parameters()

172 Chapter 21. algorithms.clustering.ggmixture
22.1 Module: `algorithms.clustering.gmm`

Inheritance diagram for `nipy.algorithms.clustering.gmm`:

```
    clustering.gmm.GMM
      |           |
      v           v
clustering.gmm.GridDescriptor
```

Gaussian Mixture Model Class: contains the basic fields and methods of GMMs. The class GMM_old uses C bindings which are computationally and memory efficient.

Author: Bertrand Thirion, 2006-2009

22.2 Classes

22.2.1 GMM

```python
class nipy.algorithms.clustering.gmm.GMM (k=1, dim=1, prec_type='full', means=None, precisions=None, weights=None)
    Bases: object

Standard GMM.

this class contains the following members:

- k (int): the number of components in the mixture
- dim (int): is the dimension of the data
- prec_type = 'full' (string) is the parameterization
  of the precisions/covariance matrices: either 'full' or 'diagonal'.

- means: array of shape (k,dim): all the means (mean parameters) of the components
- precisions: array of shape (k,dim,dim): the precisions (inverse covariance matrix) of the components
```
weights: array of shape(k): weights of the mixture

Methods

**average_log_like**(x[, tiny]) returns the averaged log-likelihood of the mode for the dataset x

**bic**(like[, tiny]) Computation of bic approximation of evidence

**check**() Checking the shape of different matrices involved in the model

**check_x**(x) essentially check that x.shape[1]==self.dim

**estimate**(x[, niter, delta, verbose]) Estimation of the model given a dataset x

**evidence**(x) Computation of bic approximation of evidence

**guess_regularizing**(x[, bcheck]) Set the regularizing priors as weakly informative

**initialize**(x) Initializes self according to a certain dataset x: 1.

**initialize_and_estimate**(x[, z, niter, ...]) Estimation of self given x

**likelihood**(x) return the likelihood of the model for the data x

**map_label**(x[, like]) return the MAP labelling of x

**mixture_likelihood**(x) Returns the likelihood of the mixture for x

**plugin**(means, precisions, weights) Set manually the weights, means and precision of the model

**pop**(like[, tiny]) compute the population, i.e. the statistics of allocation

**show**(x, gd[, density, axes]) Function to plot a GMM, still in progress

**show_components**(x, gd[, density, mpaxes]) Function to plot a GMM – Currently, works only in 1D

**test**(x[, tiny]) Returns the log-likelihood of the mixture for x

**train**(x[, z, niter, delta, ninit, verbose]) Idem initialize_and_estimate

**unweighted_likelihood**(x) return the likelihood of each data for each component

**update**(x, l) Identical to self._Mstep(x,l)

```python
__init__ (k=1, dim=1, prec_type='full', means=None, precisions=None, weights=None)
Initialize the structure, at least with the dimensions of the problem

Parameters  k (int) the number of classes of the model:
  dim (int) the dimension of the problem:

  prec_type = ‘full’: covaraince:precision parameterization
    (diagonal ‘diag’ or full ‘full’).
  means = None: array of shape (self.k,self.dim):
  precisions = None: array of shape (self.k,self.dim,self.dim):
    or (self.k, self.dim)
  weights= None: array of shape (self.k):

  By default, means, precision and weights are set as:
  zeros():
  eye():
  1/k ones():

  with the correct dimensions:

  **average_log_like**(x, tiny=1e-15)
  returns the averaged log-likelihood of the mode for the dataset x
```
Parameters  

- **x**: array of shape (n_samples, self.dim):  
  the data used in the estimation process

- **tiny = 1.e-15**: a small constant to avoid numerical singularities:

\[
\text{bic}(\text{like}, \text{tiny}=1e-15)
\]

Computation of bic approximation of evidence

- **Parameters**  
  - **like**: array of shape (n_samples, self.k):  
    component-wise likelihood

- **tiny = 1.e-15**: a small constant to avoid numerical singularities:

Returns  

- **the bic value**, float:

\[
\text{check}()
\]

Checking the shape of different matrices involved in the model

\[
\text{check_x(}x)\]

essentially check that x.shape[1]==self.dim

x is returned with possibly reshaping

\[
\text{estimate}(x, \text{niter}=100, \text{delta}=0.0001, \text{verbose}=0)
\]

Estimation of the model given a dataset x

- **Parameters**  
  - **x**: array of shape (n_samples, dim):  
    the data from which the model is estimated

- **niter = 100**: maximal number of iterations in the estimation process:

- **delta = 1.e-4**: increment of data likelihood at which convergence is declared

- **verbose = 0**: verbosity mode:

Returns  

- **bic**: an asymptotic approximation of model evidence

\[
\text{evidence}(x)
\]

Computation of bic approximation of evidence

- **Parameters**  
  - **x**: array of shape (n_samples, dim):  
    the data from which bic is computed

Returns  

- **the bic value**: 

\[
\text{guess_regularizing}(x, \text{bcheck}=1)
\]

Set the regularizing priors as weakly informative according to Fraley and raftery; Journal of Classification 24:155-181 (2007)

- **Parameters**  
  - **x**: array of shape (n_samples, dim):  
    the data used in the estimation process

\[
\text{initialize}(x)
\]

Initializes self according to a certain dataset x: 1. sets the regularizing hyper-parameters 2. initializes z using a k-means algorithm, then 3. update the parameters

- **Parameters**  
  - **x**, **array of shape (n_samples, self.dim)**:

    - the data used in the estimation process
**initialize_and_estimate** *(x, z=None, niter=100, delta=0.0001, ninit=1, verbose=0)*

Estimation of self given x

**Parameters**
- **x** array of shape (n_samples, dim): 
  the data from which the model is estimated
- **z** = None: array of shape (n_samples): 
  a prior labelling of the data to initialize the computation
- **niter**=100: maximal number of iterations in the estimation process:
- **delta** = 1.e-4: increment of data likelihood at which:
  convergence is declared
- **ninit**=1: number of initialization performed:
  to reach a good solution
- **verbose**=0: verbosity mode:

**Returns** 
the best model is returned:

**likelihood** *(x)*

return the likelihood of the model for the data x the values are weighted by the components weights

**Parameters**
- **x** array of shape (n_samples, self.dim):
  the data used in the estimation process

**Returns**
- **like** array of shape(n_samples, self.k):
  component-wise likelihood

**map_label** *(x, like=None)*

return the MAP labelling of x

**Parameters**
- **x** array of shape (n_samples, dim):
  the data under study
- **like** = None array of shape(n_samples, self.k):
  component-wise likelihood if like==None, it is recomputed

**Returns**
- **z** array of shape(n_samples): the resulting MAP labelling:
  of the rows of x

**mixture_likelihood** *(x)*

Returns the likelihood of the mixture for x

**Parameters**
- **x**: array of shape (n_samples, self.dim):
  the data used in the estimation process

**plugin** *(means, precisions, weights)*

Set manually the weights, means and precision of the model

**Parameters**
- **means**: array of shape (self.k, self.dim):
- **precisions**: array of shape (self.k, self.dim, self.dim):
  or (self.k, self.dim)
- **weights**: array of shape (self.k):
**pop** *(like, tiny=1e-15)*

compute the population, i.e. the statistics of allocation

- **Parameters**
  - `like` : array of shape `(n_samples, self.k)` :
    - the likelihood of each item being in each class

**show** *(x, gd, density=None, axes=None)*

Function to plot a GMM, still in progress Currently, works only in 1D and 2D

- **Parameters**
  - `x` : array of shape `(n_samples, dim)` :
    - the data under study
  - `gd` : GridDescriptor instance :
  - `density` : array of shape `(prod(gd.n_bins))` :
    - density of the model on the discrete grid implied by `gd` by default, this is recomputed

**show_components** *(x, gd, density=None, mpaxes=None)*

Function to plot a GMM – Currently, works only in 1D

- **Parameters**
  - `x` : array of shape `(n_samples, dim)` :
    - the data under study
  - `gd` : GridDescriptor instance :
  - `density` : array of shape `(prod(gd.n_bins))` :
    - density of the model on the discrete grid implied by `gd` by default, this is recomputed
  - `mpaxes` : axes handle to make the figure, optional, :
    - if None, a new figure is created

**test** *(x, tiny=1e-15)*

Returns the log-likelihood of the mixture for `x`

- **Parameters**
  - `x` : array of shape `(n_samples, self.dim)` :
    - the data used in the estimation process

- **Returns**
  - `ll` : array of shape `(n_samples)` :
    - the log-likelihood of the rows of `x`

**train** *(x, z=None, niter=100, delta=0.0001, ninit=1, verbose=0)*

Idem initialize_and_estimate

**unweighted_likelihood** *(x)*

return the likelihood of each data for each component the values are not weighted by the component weights

- **Parameters**
  - `x` : array of shape `(n_samples, self.dim)` :
    - the data used in the estimation process

- **Returns**
  - `like` : array of shape `(n_samples, self.k)` :
    - unweighted component-wise likelihood
Notes

Hopefully faster

**unweighted likelihood** \((x)\)

return the likelihood of each data for each component the values are not weighted by the component weights

**Parameters**  

\(x\): array of shape \((n\_samples, self\_dim)\):

the data used in the estimation process

**Returns**  

like, array of shape\((n\_samples, self\_k)\):

unweighted component-wise likelihood

**update** \((x, l)\)

Identical to self\._Mstep(x, l)

### 22.2.2 GridDescriptor

class nipy.algorithms.clustering.gmm.GridDescriptor \((dim=1, lim=None, n\_bins=None)\)

**Bases:** object

A tiny class to handle cartesian grids

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>make_grid()</code></td>
<td>Compute the grid points</td>
</tr>
<tr>
<td><code>set(lim[, n\_bins])</code></td>
<td>set the limits of the grid and the number of bins</td>
</tr>
</tbody>
</table>

**__init__** \((dim=1, lim=None, n\_bins=None)\)

**Parameters**

\(dim\): int, optional, :

the dimension of the grid

\(lim\): list of len\((2*\text{self.dim})\), :

the limits of the grid as \((xmin, xmax, ymin, ymax, ...)\)

\(n\_bins\): list of len\((\text{self.dim})\), :

the number of bins in each direction

**make_grid()**

Compute the grid points

**Returns**  

grid: array of shape \((nb\_nodes, self\_dim)\):

where \(nb\_nodes\) is the prod of \(self\_n\_bins\)

**set** \((lim, n\_bins=10)\)

set the limits of the grid and the number of bins

**Parameters**

\(lim\): list of len\((2*\text{self.dim})\), :

the limits of the grid as \((xmin, xmax, ymin, ymax, ...)\)

\(n\_bins\): list of len\((\text{self.dim})\), optional :
22.3 Functions

`nipy.algorithms.clustering.gmm.best_fitting_GMM(x, krange, prec_type='full', niter=100, delta=0.0001, ninit=1, verbose=0)`

Given a certain dataset `x`, find the best-fitting GMM with a number `k` of classes in a certain range defined by `krange`.

**Parameters**
- `x`: array of shape `(n_samples, dim)`, the data from which the model is estimated
- `krange`: list of floats, the range of values to test for `k`
- `prec_type`: string (to be chosen within ‘full’,’diag’), optional, the covariance parameterization
- `niter`: int, optional, maximal number of iterations in the estimation process
- `delta`: float, optional, increment of data likelihood at which convergence is declared
- `ninit`: int
- `verbose=0`: verbosity mode

**Returns**
- `mg`: the best-fitting GMM instance

`nipy.algorithms.clustering.gmm.plot2D(x, my_gmm, z=None, with_dots=True, log_scale=False, mpaxes=None, verbose=0)`

Given a set of points in a plane and a GMM, plot them.

**Parameters**
- `x`: array of shape `(npoints, dim=2)`, sample points
- `my_gmm`: GMM instance, whose density has to be plotted
- `z`: array of shape `(npoints)`, optional, that gives a labelling of the points in `x` by default, it is not taken into account
- `with_dots`, bool, optional
- `log_scale`, bool, optional
- `mpaxes`=None, int, optional
- `verbose`: verbosity mode, optional
Returns `gd`, `GridDescriptor` instance,:

that represents the grid used in the function

`ax`, handle to the figure axes:

Notes

`my_gmm` is assumed to have have a `nixture_likelihood` method that takes an array of points of shape `(np, dim)` and returns an array of shape `(np, my_gmm.k)` that represents the likelihood component-wise.
23.1 Module: `algorithms.clustering.hierarchical_clustering`

Inheritance diagram for `nipy.algorithms.clustering.hierarchical_clustering`:

![Inheritance diagram]

These routines perform some hierarchical agglomerative clustering of some input data. The following alternatives are proposed: - Distance based average-link - Similarity-based average-link - Distance based maximum-link - Ward’s algorithm under graph constraints - Ward’s algorithm without graph constraints

In this latest version, the results are returned in a ‘WeightedForest’ structure, which gives access to the clustering hierarchy, facilitates the plot of the result etc.

For back-compatibility, *_segment versions of the algorithms have been appended, with the old API (except the qmax parameter, which now represents the number of wanted clusters)

Author : Bertrand Thirion, Pamela Guevara, 2006-2009

23.2 Class

23.3 WeightedForest

```python
class nipy.algorithms.clustering.hierarchical_clustering.WeightedForest(V, parents=None, height=None):
    Bases: nipy.algorithms.graph.forest.Forest

This is a weighted Forest structure, i.e. a tree - each node has one parent and children (hierarchical structure) - some of the nodes can be viewed as leaves, other as roots - the edges within a tree are associated with a weight: +1 from child to parent -1 from parent to child - additionally, the nodes have a value, which is called ‘height’, especially useful from dendrograms
```
Methods

- `adjacency()` returns the adjacency matrix of the graph as a sparse coo matrix.
- `all_distances(seed)` returns all the distances of the graph as a tree.
- `anti_symmeterize()` anti-symmetrize self, i.e. produces the graph.
- `cc()` Compte the different connected components of the graph.
- `check()` Check that self is indeed a forest, i.e.
- `check_compatible_height()` Check that height[parents[i]]>=height[i] for all nodes.
- `cliques()` Extraction of the graphe cliques.
- `compact_neighb()` returns a compact representation of self.
- `compute_children()` Define the children of each node (stored in self.children).
- `copy()` returns a copy of self.
- `cut_redundancies()` Returns a graph with redundant edges removed: each edge (ab) is present ony once in the.
- `define_graph_attributes()` Returns a graph with redundant edges removed: each edge (ab) is present ony once in the
- `degrees()` Returns the degree of the graph vertices.
- `depth_from_leaves()` compute an index for each node: 0 for the leaves, 1 for
- `dijkstra(seed)` Returns all the [graph] geodesic distances starting from seed.
- `floyd(seed)` Compute all the geodesic distances starting from seeds.
- `from_3d_grid(xyz, k)` Sets the graph to be the topological neighbours graph.
- `get_E()` To get the number of edges in the graph.
- `get_V()` To get the number of vertices in the graph.
- `get_children(v)` Get the children of a node/each node.
- `get_descendants(v, exclude_self)` returns the nodes that are children of v as a list.
- `get_edges()` To get the graph’s edges.
- `get_height()` Get the height array.
- `get_vertices()` To get the graph’s vertices (as id).
- `is_connected()` States whether self is connected or not.
- `isleaf()` Identification of the leaves of the forest.
- `isroot()` Returns an indicator of nodes being roots.
- `kruskal()` Creates the Minimum Spanning Tree of self using Kruskal’s algo.
- `leaves_of_a_subtree(ids, custom)` tests whether the given nodes are the leaves of a certain subtree.
- `left_incidence()` Return left incidence matrix.
- `list_of_neighbors()` returns the set of neighbors of self as a list of arrays.
- `list_of_subtrees()` returns the list of all non-trivial subtrees in the graph.
- `main_cc()` Returns the indexes of the vertices within the main cc.
- `merge_simple_branches()` Return a subforest, where chained branches are collapsed.
- `normalize(c)` Normalize the graph according to the index c.
- `partition(threshold)` Partition the tree according to a cut criterion.
- `plot(ax)` Plot the dendrogram associated with self.
- `plot_height()` Plot the height of the non-leaves nodes.
- `propagate_upward(label)` Propagation of a certain labelling from leaves to roots.
- `propagate_upward_and(prop)` propagates from leaves to roots some binary property of the nodes.
- `remove_edges(valid)` Removes all the edges for which valid==0.
- `remove_trivial_edges()` Removes trivial edges, i.e.
- `reorder_from_leaves_to_roots()` reorder the tree so that the leaves come first then their.
- `right_incidence()` Return right incidence matrix.
- `set_edges(edges)` Sets the graph’s edges.
- `set_euclidian(X)` Compute the weights of the graph as the distances between the
- `set_gaussian(X, sigma)` Compute the weights of the graph as a gaussian function.
Table 23.1 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>set_height([height])</td>
<td>Set the height array</td>
</tr>
<tr>
<td>set_weights(weights)</td>
<td>Set edge weights</td>
</tr>
<tr>
<td>show([X, ax])</td>
<td>Plots the current graph in 2D</td>
</tr>
<tr>
<td>split(k)</td>
<td>idem as partition, but a number of components are supplied instead</td>
</tr>
<tr>
<td>subforest(valid)</td>
<td>Creates a subforest with the vertices for which valid &gt; 0</td>
</tr>
<tr>
<td>subgraph(valid)</td>
<td>Creates a subgraph with the vertices for which valid &gt; 0</td>
</tr>
<tr>
<td>symmeterize()</td>
<td>Symmeterize self, modify edges and weights so that</td>
</tr>
<tr>
<td>to_coo_matrix()</td>
<td>Return adjacency matrix as coo sparse</td>
</tr>
<tr>
<td>tree_depth()</td>
<td>Returns the number of hierarchical levels in the tree</td>
</tr>
<tr>
<td>voronoi_diagram(seeds, samples)</td>
<td>Defines the graph as the Voronoi diagram (VD)</td>
</tr>
<tr>
<td>voronoi_labelling(seed)</td>
<td>Performs a voronoi labelling of the graph</td>
</tr>
</tbody>
</table>

```python
__init__(V, parents=None, height=None)
```

Parameters

- **V**: the number of edges of the graph:
  - **parents=None**: array of shape (V):
    - the parents of the graph by default, the parents are set to range(V), i.e. each node is its own parent, and each node is a tree
  - **height=None**: array of shape(V):
    - the height of the nodes

Returns

- **adj**: scipy.sparse matrix instance,
  - that encodes the adjacency matrix of self

```python
all_distances (seed=None)
```

Parameters

- **seed=None**: array of shape(nbseed) with values in [0..self.V-1]:
  - set of vertices from which the distances are computed

Returns

- **dg**: array of shape(nseed, self.V), the resulting distances:

Notes

By convention infinite distances are given the distance np.inf

```python
anti_symmeterize()
```

anti-symmeterize self, i.e. produces the graph whose adjacency matrix would be the antisymmetric part of its current adjacency matrix

```python
cc()
```

Compte the different connected components of the graph.

Returns

- **label**: array of shape(self.V), labelling of the vertices:

```python
check()
```

Check that self is indeed a forest, i.e. contains no loop

Returns

- a boolean b=0 iff there are loops, 1 otherwise:
Notes

Slow implementation, might be rewritten in C or cython

```python
check_compatible_height()
```
Check that height[parents[i]]>=height[i] for all nodes

```python
clques()
```
Extraction of the graph cliques these are defined using replicator dynamics equations

Returns cliques: array of shape (self.V), type (np.int):
labelling of the vertices according to the clique they belong to

```python
compact_neighb()
```
returns a compact representation of self

Returns idx: array of of shape(self.V + 1):
the positions where to find the neighbors of each node within neigh and weights

```python
neigh: array of shape(self.E), concatenated list of neighbors:
weights: array of shape(self.E), concatenated list of weights:
```

```python
compute_children()
```
Define the children of each node (stored in self.children)

```python
copy()
```
returns a copy of self

```python
cut_redundancies()
```
Returns a graph with redundant edges removed: each edge (ab) is present only once in the edge matrix: the corresponding weights are added.

Returns the resulting WeightedGraph:

```python
define_graph_attributes()
```
define the edge and weights array

```python
degrees()
```
Returns the degree of the graph vertices.

Returns rdegree: (array, type=int, shape=(self.V)), the right degrees:

ldegree: (array, type=int, shape=(self.V)), the left degrees:

```python
depth_from_leaves()
```
compute an index for each node: 0 for the leaves, 1 for their parents etc. and maximal for the roots.

Returns depth: array of shape (self.V): the depth values of the vertices:

```python
dijkstra(seed=0)
```
Returns all the [graph] geodesic distances starting from seed x Parameters ———- seed (int, >-1, <self.V)
or array of shape(p)
edge(s) from which the distances are computed

Returns dg: array of shape (self.V),:
the graph distance dg from ant vertex to the nearest seed
Notes

It is mandatory that the graph weights are non-negative.

```
floyd(seed=None)
    Compute all the geodesic distances starting from seeds
    Parameters  seed= None: array of shape (nbseed), type np.int :
        vertex indexes from which the distances are computed if seed==None, then every edge
        is a seed point
    Returns  dg array of shape (nbseed, self.V) :
        the graph distance dg from each seed to any vertex
```

Notes

It is mandatory that the graph weights are non-negative. The algorithm proceeds by repeating Dijkstra’s
algo for each seed. Floyd’s algo is not used (O(self.V)^3 complexity...)

```
from_3d_grid(xyz, k=18)
    Sets the graph to be the topological neighbours graph of the three-dimensional coordinates set xyz, in the
    k-connectivity scheme
    Parameters  xyz: array of shape (self.V, 3) and type np.int, :
        k = 18: the number of neighbours considered. (6, 18 or 26) :
    Returns  E(int): the number of edges of self :
```

```
get_E()
    To get the number of edges in the graph

get_V()
    To get the number of vertices in the graph

get_children(v=-1)
    Get the children of a node/each node
    Parameters  v: int, optional :
        a node index
    Returns  children: list of int the list of children of node v (if v is provided) :
        a list of lists of int, the children of all nodes otherwise

get_descendants(v, exclude_self=False)
    returns the nodes that are children of v as a list
    Parameters  v: int, a node index :
    Returns  desc: list of int, the list of all descendant of the input node :
```

```
get_edges()
    To get the graph’s edges

get_height()
    Get the height array

get_vertices()
    To get the graph’s vertices (as id)
```

23.3. WeightedForest
get_weights()

is_connected()
States whether self is connected or not

isleaf()
Identification of the leaves of the forest

    Returns leaves: bool array of shape(self.V), indicator of the forest’s leaves:

isroot()
Returns an indicator of nodes being roots

    Returns roots, array of shape(self.V, bool), indicator of the forest’s roots:

kruskal()
Creates the Minimum Spanning Tree of self using Kruskal’s algo. efficient is self is sparse

    Returns K, WeightedGraph instance: the resulting MST:

Notes
If self contains several connected components, will have the same number k of connected components

leaves_of_a_subtree (ids, custom=False)
tests whether the given nodes are the leaves of a certain subtree

    Parameters ids: array of shape (n) that takes values in [0..self.V-1]:
        custom == False, boolean:
            if custom==true the behavior of the function is more specific - the different connected components are considered as being in a same greater tree - when a node has more than two subbranches, any subset of these children is considered as a subtree

left_incidence()
Return left incidence matrix

    Returns left_incid: list:
        the left incidence matrix of self as a list of lists: i.e. the list[[e.0.0, .., e.0.i(0)], .., [e.V.0, E.V.i(V)]] where e.i.j is the set of edge indexes so that e.i.j[0] = i

list_of_neighbors()
returns the set of neighbors of self as a list of arrays

list_of_subtrees()
returns the list of all non-trivial subtrees in the graph Caveat: theis function assumes that the vertices are sorted in a way such that parent[i]>i forall i Only the leaves are listeed, not the subtrees themselves

main_cc()
Returns the indexes of the vertices within the main cc

    Returns idx: array of shape (sizeof main cc):

merge_simple_branches()
Return a subforest, where chained branches are collapsed

    Returns sf, Forest instance, same as self, without any chain:

normalize (c=0)
Normalize the graph according to the index c Normalization means that the sum of the edges values that go into or out each vertex must sum to 1
Parameters  
\( c=0 \) in \{0, 1, 2\}, optional: index that designates the way:

- \( c == 0 \) \( \Rightarrow \) for each vertex \( a \), \( \sum\{\text{edge}[e, 0]=a\} \ \text{D}[e]=1 \)
- \( c == 1 \) \( \Rightarrow \) for each vertex \( b \), \( \sum\{\text{edge}[e, 1]=b\} \ \text{D}[e]=1 \)
- \( c == 2 \) \( \Rightarrow \) symmetric (‘l2’) normalization

Notes

Note that when \( \sum\{\text{edge}[e, .] == a \} \ \text{D}[e] = 0 \), nothing is performed.

**partition** *(threshold)*
Partition the tree according to a cut criterion.

**plot** *(ax=None)*
Plot the dendrogram associated with self the rank of the data in the dendogram is returned.

- **Parameters**  
  \( ax: \) axis handle, optional:

- **Returns**  
  \( ax, \) the axis handle:

**plot_height** *
Plot the height of the non-leaves nodes.

**propagate_upward** *(label)*
Propagation of a certain labelling from leaves to roots Assuming that label is a certain positive integer field this propagates these labels to the parents whenever the children nodes have coherent properties otherwise the parent value is unchanged.

- **Parameters**  
  \( label: \) array of shape(self.V):

- **Returns**  
  \( label: \) array of shape(self.V):

**propagate_upward_and** *(prop)*
propagates from leaves to roots some binary property of the nodes so that prop[parents] = logical_and(prop[children])

- **Parameters**  
  \( prop, \) array of shape(self.V), the input property:

- **Returns**  
  \( prop, \) array of shape(self.V), the output property field:

**remove_edges** *(valid)*
Removes all the edges for which valid==0

- **Parameters**  
  \( valid: \) (self.E,) array

**remove_trivial_edges** *
Removes trivial edges, i.e. edges that are (vv)-like self.weights and self.E are corrected accordingly.

- **Returns**  
  \( self.E (int): \) The number of edges:

**reorder_from_leaves_to_roots** *
reorder the tree so that the leaves come first then their parents and so on, and the roots are last.

- **Returns**  
  \( order: \) array of shape(self.V):

  the order of the old vertices in the reordered graph

**right_incidence** *
Return right incidence matrix

- **Returns**  
  \( right_incid: \) list:

  the right incidence matrix of self as a list of lists: i.e. the list[[e.0.0, ..., e.0.i(0)], ..., [e.V.0, E.V.i(V)]] where e.i.j is the set of edge indexes so that e.i.j[1] = i
set_edges \((edges)\)

Sets the graph’s edges

set_euclidian \((X)\)

Compute the weights of the graph as the distances between the corresponding rows of \(X\), which represents an embedding of self

Parameters

\(X\) array of shape \((self.V, edim)\):
the coordinate matrix of the embedding

set_gaussian \((X, \sigma=0)\)

Compute the weights of the graph as a gaussian function of the distance between the corresponding rows of \(X\), which represents an embedding of self

Parameters

\(X\) array of shape \((self.V, dim)\):
the coordinate matrix of the embedding

\(\sigma=0\), float: the parameter of the gaussian function:

Notes

When \(\sigma == 0\), the following value is used:

\[
\sigma = \text{sqrt} \left( \text{mean} (||X[\text{self.edges[:, 0]}, :] - X[\text{self.edges[:, 1]}, :])^2) \right)
\]

set_height \((height=None)\)

Set the height array

set_weights \((weights)\)

Set edge weights

Parameters

\(weights\): array:
array shape \((self.V)\): edges weights

show \((X=None, ax=None)\)

Plots the current graph in 2D

Parameters

\(X\) : None or array of shape \((self.V, 2)\)
a set of coordinates that can be used to embed the vertices in 2D. If \(X\).shape[1]>2, a svd reduces \(X\) for display. By default, the graph is presented on a circle

\(ax\) : None or int, optional:
ax handle

Returns

\(ax\): axis handle:

Notes

This should be used only for small graphs.

split \((k)\)

idem as partition, but a number of components are supplied instead

subforest \((valid)\)

Creates a subforest with the vertices for which \(valid > 0\)

Parameters

\(valid\): array of shape \((self.V)\): indicator of the selected nodes:

Returns

\(subforest\): a new forest instance, with a reduced set of nodes:
Notes

The children of deleted vertices become their own parent

**subgraph** *(valid)*

Creates a subgraph with the vertices for which valid>0 and with the corresponding set of edges

**Parameters**

- **valid**: array of shape (self.V): nonzero for vertices to be retained

**Returns**

- **G**: WeightedGraph instance, the desired subgraph of self

Notes

The vertices are renumbered as [1..p] where p = sum(valid>0) when sum(valid==0) then None is returned

**symmeterize** ()

Symmeterize self, modify edges and weights so that self.adjacency becomes the symmetric part of the current self.adjacency.

**to_coo_matrix** ()

Return adjacency matrix as coo sparse

**Returns**

- **sp**: scipy.sparse matrix instance: that encodes the adjacency matrix of self

**tree_depth** ()

Returns the number of hierarchical levels in the tree

**voronoi_diagram** *(seeds, samples)*

Defines the graph as the Voronoi diagram (VD) that links the seeds. The VD is defined using the sample points.

**Parameters**

- **seeds**: array of shape (self.V, dim):
- **samples**: array of shape (nsamples, dim):

Notes

By default, the weights are a Gaussian function of the distance The implementation is not optimal

**voronoi_labelling** *(seed)*

Performs a voronoi labelling of the graph

**Parameters**

- **seed**: array of shape (nseeds), type (np.int),:

**Returns**

- **labels**: array of shape (self.V) the labelling of the vertices

23.4 Functions

**nipy.algorithms.clustering.hierarchical_clustering.average_link_graph** *(G)*

Agglomerative function based on a (hopefully sparse) similarity graph

**Parameters**

- **G**: the input graph:

**Returns**

- **t**: a weightForest structure that represents the dendrogram of the data:
Agglomerative function based on a (hopefully sparse) similarity graph

**Parameters**

- **G** the input graph:
  - `stop` float: the stopping criterion
  - `qmax` int, optional: the number of desired clusters (in the limit of the stopping criterion)
  - `verbose` bool, optional: If True, print diagnostic information

**Returns**

- **u**: array of shape (G.V):
  - a labelling of the graph vertices according to the criterion
- **cost**: array of shape (G.V(?)):
  - the cost of each merge step during the clustering procedure

The similarity values are weighted averaged, where pop[i] and pop[j] yield the relative weights. This is used in `average_link_slow` (deprecated)

Agglomerative function based on a topology-defining graph and a feature matrix.

**Parameters**

- **G**: graph
  - the input graph (a topological graph essentially)
- **feature**: array of shape (G.V,dim_feature)
  - vectorial information related to the graph vertices
- **verbose** bool, optional: If True, print diagnostic information

**Returns**

- **t**: `WeightedForest` instance
  - structure that represents the dendrogram

**Notes**

When G has more than 1 connected component, t is no longer a tree. This case is handled cleanly now
nipy.algorithms.clustering.hierarchical_clustering.ward_field_segment \( (F, \)
\[
\text{stop=-} \quad 1, \\
\text{qmax=-} \quad 1, \\
\text{verbose=False})
\]

Agglomerative function based on a field structure

Parameters F the input field (graph+feature):
- stop: float, optional
  the stopping criterion. If stop=-1, then no stopping criterion is used
- qmax: int, optional
  the maximum number of desired clusters (in the limit of the stopping criterion)
- verbose: bool, optional
  If True, print diagnostic information

Returns u: array of shape (F.V):
- labelling of the graph vertices according to the criterion
- cost array of shape (F.V - 1):
  the cost of each merge step during the clustering procedure

Notes

See ward_quick_segment for more information

Caveat: only approximate

nipy.algorithms.clustering.hierarchical_clustering.ward_quick \( (G, \text{feature, } \) 
\[
\text{verbose=False})
\]

Agglomerative function based on a topology-defining graph and a feature matrix.

Parameters G: graph instance
- topology-defining graph
- feature: array of shape (G.V,dim_feature):
  some vectorial information related to the graph vertices
- verbose: bool, optional
  If True, print diagnostic information

Returns t: weightForest instance,:
- that represents the dendrogram of the data

Notes:

---

Hopefully a quicker version:

A euclidean distance is used in the feature space:
Caveat: only approximate
Agglomerative function based on a topology-defining graph and a feature matrix.

Parameters

$G$: labs.graph.WeightedGraph instance:
the input graph (a topological graph essentially)

feature array of shape $(G.V, \text{dim\_feature})$:
vectorial information related to the graph vertices

$\text{stop1}$: int or float, optional
the stopping criterion if $\text{stop}=-1$, then no stopping criterion is used

$qmax$ : int, optional
the maximum number of desired clusters (in the limit of the stopping criterion)

verbose : bool, optional
If True, print diagnostic information

Returns

$u$: array of shape $(G.V)$:
labelling of the graph vertices according to the criterion

cost: array of shape $(G.V - 1)$:
the cost of each merge step during the clustering procedure

Notes

Hopefully a quicker version
A euclidean distance is used in the feature space
Caveat : only approximate

Agglomerative function based on a topology-defining graph and a feature matrix.

Parameters

$G$: graph object
the input graph (a topological graph essentially)

feature : array of shape $(G.V, \text{dim\_feature})$
some vectorial information related to the graph vertices

stop : int or float, optional
the stopping criterion. if $\text{stop}=-1$, then no stopping criterion is used

qmax : int, optional
the maximum number of desired clusters (in the limit of the stopping criterion)

**verbose**: bool, optional

If True, print diagnostic information

**Returns**

- **u**: array of shape (G.V):
  - a labelling of the graph vertices according to the criterion

- **cost**: array of shape (G.V - 1):
  - the cost of each merge step during the clustering procedure

**Notes**

A euclidean distance is used in the feature space

Caveat: when the number of cc in G (nbcc) is greater than qmax, u contains nbcc values, not qmax!
24.1 Module: algorithms.clustering.imm

Inheritance diagram for nipy.algorithms.clustering.imm:

```
```

Infinite mixture model: A generalization of Bayesian mixture models with an unspecified number of classes

24.2 Classes

24.2.1 IMM

```
class nipy.algorithms.clustering.imm.IMM(alpha=0.5, dim=1)
    Bases: nipy.algorithms.clustering.bgmm.BGMM
```

The class implements Infinite Gaussian Mixture model or Dirichlet Process Mixture Model. This simply a generalization of Bayesian Gaussian Mixture Models with an unknown number of classes.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>average_log_like(x[, tiny])</code></td>
<td>returns the averaged log-likelihood of the mode for the dataset x</td>
</tr>
<tr>
<td><code>bayes_factor(x, z[, nperm, verbose])</code></td>
<td>Evaluate the Bayes Factor of the current model using Chib’s method</td>
</tr>
<tr>
<td><code>bic(like[, tiny])</code></td>
<td>Computation of bic approximation of evidence</td>
</tr>
<tr>
<td><code>check()</code></td>
<td>Checking the shape of different matrices involved in the model</td>
</tr>
<tr>
<td><code>check_x(x)</code></td>
<td>essentially check that x.shape[1]==self.dim</td>
</tr>
<tr>
<td><code>conditional_posterior_proba(x, z[, perm])</code></td>
<td>Compute the probability of the current parameters of self</td>
</tr>
<tr>
<td><code>cross_validated_update(x, z, plike[, kfold])</code></td>
<td>This is a step in the sampling procedure</td>
</tr>
<tr>
<td><code>estimate(x[, niter, delta, verbose])</code></td>
<td>Estimation of the model given a dataset x</td>
</tr>
<tr>
<td><code>evidence(x, z[, nperm, verbose])</code></td>
<td>See bayes_factor(self, x, z, nperm=0, verbose=0)</td>
</tr>
<tr>
<td><code>guess_priors(x[, nocheck])</code></td>
<td>Set the priors in order of having them weakly uninformative</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 24.1 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>guess_regularizing(x[, bcheck])</code></td>
<td>Set the regularizing priors as weakly informative</td>
</tr>
<tr>
<td><code>initialize(x)</code></td>
<td>initialize z using a k-means algorithm, then update the parameters</td>
</tr>
<tr>
<td><code>initialize_and_estimate(x[, z, niter, ...])</code></td>
<td>Estimation of self given x</td>
</tr>
<tr>
<td><code>likelihood(x[, plike])</code></td>
<td>return the likelihood of the model for the data x</td>
</tr>
<tr>
<td><code>likelihood_under_the_prior(x)</code></td>
<td>Computes the likelihood of x under the prior</td>
</tr>
<tr>
<td><code>map_label(x[, like])</code></td>
<td>return the MAP labelling of x</td>
</tr>
<tr>
<td><code>mixture_likelihood(x)</code></td>
<td>Returns the likelihood of the mixture for x</td>
</tr>
<tr>
<td><code>plugin(means, precisions, weights)</code></td>
<td>Set manually the weights, means and precision of the model</td>
</tr>
<tr>
<td><code>pop(z)</code></td>
<td>compute the population, i.e. the statistics of allocation</td>
</tr>
<tr>
<td><code>probability_under_prior()</code></td>
<td>Compute the probability of the current parameters of self</td>
</tr>
<tr>
<td><code>reduce(z)</code></td>
<td>Reduce the assignments by removing empty clusters and update self.k</td>
</tr>
<tr>
<td><code>sample(x[, niter, sampling_points, init, ...])</code></td>
<td>sample the indicator and parameters</td>
</tr>
<tr>
<td><code>sample_and_average(x[, niter, verbose])</code></td>
<td>sample the indicator and parameters</td>
</tr>
<tr>
<td><code>sample_indicator(like)</code></td>
<td>Sample the indicator from the likelihood</td>
</tr>
<tr>
<td><code>set_constant_densities([prior_dens])</code></td>
<td>Set the null and prior densities as constant</td>
</tr>
<tr>
<td><code>set_priors(x)</code></td>
<td>Set the priors in order of having them weakly uninformative</td>
</tr>
<tr>
<td><code>show(x, gd[, density, axes])</code></td>
<td>Function to plot a GMM, still in progress</td>
</tr>
<tr>
<td><code>show_components(x, gd[, density, mpaxes])</code></td>
<td>Function to plot a GMM – Currently, works only in 1D</td>
</tr>
<tr>
<td><code>simple_update(x, z, plike)</code></td>
<td>This is a step in the sampling procedure</td>
</tr>
<tr>
<td><code>test(x[, tiny])</code></td>
<td>Returns the log-likelihood of the mixture for x</td>
</tr>
<tr>
<td><code>train(x[, z, niter, delta, ninit, verbose])</code></td>
<td>Idem initialize_and_estimate</td>
</tr>
<tr>
<td><code>unweighted_likelihood(x)</code></td>
<td>return the likelihood of each data for each component</td>
</tr>
<tr>
<td><code>unweighted_likelihood_(x)</code></td>
<td>return the likelihood of each data for each component</td>
</tr>
<tr>
<td><code>update(x, z)</code></td>
<td>Update function (draw a sample of the IMM parameters)</td>
</tr>
<tr>
<td><code>update_means(x, z)</code></td>
<td>Given the allocation vector z</td>
</tr>
<tr>
<td><code>update_precisions(x, z)</code></td>
<td>Given the allocation vector z</td>
</tr>
<tr>
<td><code>update_weights(z)</code></td>
<td>Given the allocation vector z, resmaple the weights parameter</td>
</tr>
</tbody>
</table>

### __init__(alpha=0.5, dim=1)

**Parameters**

- **alpha**: float, optional, :
  - the parameter for cluster creation
- **dim**: int, optional, :
  - the dimension of the data

**Note:** use the function set_priors() to set adapted priors:

- **average_log_like (x, tiny=1e-15)**
  - returns the averaged log-likelihood of the mode for the dataset x

**Parameters**

- **x**: array of shape (n_samples,self.dim) :
  - the data used in the estimation process

**tiny** = 1.e-15: a small constant to avoid numerical singularities:

- **bayes_factor (x, z, nperm=0, verbose=0)**
  - Evaluate the Bayes Factor of the current model using Chib’s method

**Parameters**

- **x**: array of shape (nb_samples,dim) :
  - the data from which bic is computed
- **z**: array of shape (nb_samples), type = np.int :
the corresponding classification

**nperm=0: int** :
the number of permutations to sample to model the label switching issue in the computation of the Bayes Factor By default, exhaustive permutations are used

**verbose=0: verbosity mode** :

**Returns**  
bf (float) the computed evidence (Bayes factor):

**Notes**


**bic**(like, tiny=1e-15)
Computation of bic approximation of evidence

**Parameters**  
like, array of shape (n_samples, self.k) :
component-wise likelihood

tiny=1.e-15, a small constant to avoid numerical singularities :

**Returns**  
the bic value, float :

**check** ()
Checking the shape of different matrices involved in the model

**check_x**(x)
essentially check that x.shape[1]==self.dim

x is returned with possibly reshaping

**conditional_posterior_proba**(x, z, perm=None)
Compute the probability of the current parameters of self given x and z

**Parameters**  
x: array of shape (nb_samples, dim), :
the data from which bic is computed

z: array of shape (nb_samples), type = np.int, :
the corresponding classification

perm: array ok shape(nperm, self.k),typ=np.int, optional :
all permutation of z under which things will be recomputed By default, no permutation is performed

**cross_validated_update**(x, z, plike, kfold=10)
This is a step in the sampling procedure that uses internal cross-validation

**Parameters**  
x: array of shape(n_samples, dim), :
the input data

z: array of shape(n_samples), :
the associated membership variables

plike: array of shape(n_samples), :
the likelihood under the prior

24.2. Classes
Neuroimaging in Python Documentation, Release 0.3.0

kfold: int, or array of shape(n_samples), optional,

folds in the cross-validation loop

Returns like: array od shape(n_samples),

the (cross-validated) likelihood of the data

\texttt{estimate}(x, niter=100, delta=0.0001, verbose=0)

Estimation of the model given a dataset \(x\)

Parameters \(x\) array of shape (n_samples,dim):

the data from which the model is estimated

niter=100: maximal number of iterations in the estimation process:

delta = 1.e-4: increment of data likelihood at which:

convergence is declared

verbose=0: verbosity mode:

Returns bic:
an asymptotic approximation of model evidence

evidence(x, z, nperm=0, verbose=0)

See bayes_factor(self, x, z, nperm=0, verbose=0)

guess priors(x, nocheck=0)

Set the priors in order of having them weakly uninformative this is from Fraley and raftery; Journal of Classification 24:155-181 (2007)

Parameters \(x\), array of shape (nb_samples,self.dim):

the data used in the estimation process

nocheck: boolean, optional,

if nocheck==True, check is skipped

guess regularizing(x, bcheck=1)

Set the regularizing priors as weakly informative according to Fraley and raftery; Journal of Classification 24:155-181 (2007)

Parameters \(x\) array of shape (n_samples,dim):

the data used in the estimation process

initialize(x)

initialize z using a k-means algorithm, then update the parameters

Parameters \(x\): array of shape (nb_samples,self.dim):

the data used in the estimation process

initialize_and_estimate(x, z=None, niter=100, delta=0.0001, ninit=1, verbose=0)

Estimation of self given \(x\)

Parameters \(x\) array of shape (n_samples,dim):

the data from which the model is estimated

\(z = \text{None}: \text{array of shape (n_samples)}\):

a prior labelling of the data to initialize the computation

niter=100: maximal number of iterations in the estimation process:

delta = 1.e-4: increment of data likelihood at which:
convergence is declared

ninit=1: number of initialization performed:

to reach a good solution

verbose=0: verbosity mode:

Returns the best model is returned:

likelihood(x, plike=None)

return the likelihood of the model for the data x the values are weighted by the components weights

Parameters x: array of shape (n_samples, self.dim):

the data used in the estimation process

plike: array os shape (n_samples), optional x:

the desnity of each point under the prior

Returns like, array of shape(nbitem,self.k):

component-wise likelihood:

likelihood_under_the_prior(x)

Computes the likelihood of x under the prior

Parameters x, array of shape (self.n_samples,self.dim):

Returns w, the likelihood of x under the prior model (unweighted):

map_label(x, like=None)

return the MAP labelling of x

Parameters x array of shape (n_samples,dim):

the data under study

like=None array of shape(n_samples,self.k):

component-wise likelihood if like==None, it is recomputed

Returns z: array of shape(n_samples): the resulting MAP labelling:

of the rows of x

mixture_likelihood(x)

Returns the likelihood of the mixture for x

Parameters x: array of shape (n_samples, self.dim):

the data used in the estimation process

plugin(means, precisions, weights)

Set manually the weights, means and precision of the model

Parameters means: array of shape (self.k,self.dim):

precisions: array of shape (self.k,self.dim,self.dim):

or (self.k, self.dim)

weights: array of shape (self.k):

pop(z)

compute the population, i.e. the statistics of allocation

Parameters z array of shape (nb_samples), type = np.int:
the allocation variable

**Returns**  
`hist`: array shape `(self.k)` count variable

**probability_under_prior()**
Compute the probability of the current parameters of `self` given the priors

**reduce(z)**
Reduce the assignments by removing empty clusters and update `self.k`

**Parameters**  
`z`: array of shape *(n)*,

- a vector of membership variables changed in place

**Returns**  
`z`: the remapped values:

**sample**(x, `niter=1`, `sampling_points=None`, `init=False`, `kfold=None`, `verbose=0)
Sample the indicator and parameters

**Parameters**  
`x`: array of shape *(n_samples, self.dim)*,

- the data used in the estimation process

`niter`: int,

- the number of iterations to perform

**sampling_points**: array of shape *(nbpoints, self.dim)*, optional:

- points where the likelihood will be sampled this defaults to `x`

**kfold**: int or array, optional,

- parameter of cross-validation control by default, no cross-validation is used the procedure is faster but less accurate

**verbose=0**: verbosity mode:

**Returns**  
`likelihood`: array of shape *(nbpoints)*,

- total likelihood of the model

**sample_and_average**(x, `niter=1`, `verbose=0)
Sample the indicator and parameters the average values for weights, means, precisions are returned

**Parameters**  
`x = array of shape (nb_samples, dim)*

- the data from which bic is computed

`niter=1`: number of iterations:

**Returns**  
`weights`: array of shape `(self.k)`,

`means`: array of shape `(self.k, self.dim)`,

`precisions`: array of shape `(self.k, self.dim, self.dim)`:

- or `(self.k, self.dim)` these are the average parameters across samplings

**Notes**

All this makes sense only if no label switching as occurred so this is wrong in general (asymptotically).

fix: implement a permutation procedure for components identification

**sample_indicator**(like)
Sample the indicator from the likelihood
Parameters like: array of shape (nbitem, self.k):
    component-wise likelihood

Returns z: array of shape(nbitem): a draw of the membership variable:

Notes

The behaviour is different from standard bgmm in that z can take arbitrary values

set_constant_densities (prior_dens=None)
Set the null and prior densities as constant (assuming a compact domain)

Parameters prior_dens: float, optional:
    constant for the prior density

set_priors(x)
Set the priors in order of having them weakly uninformative this is from Fraley and raftery; Journal of

Parameters x, array of shape (n_samples, self.dim):
    the data used in the estimation process

show (x, gd, density=None, axes=None)
Function to plot a GMM, still in progress Currently, works only in 1D and 2D

Parameters x: array of shape(n_samples, dim):
    the data under study
gd: GridDescriptor instance:
density: array os shape(prod(gd.n_bins)):
    density of the model one the discrete grid implied by gd by default, this is recomputed

show_components (x, gd, density=None, mpaxes=None)
Function to plot a GMM – Currently, works only in 1D

Parameters x: array of shape(n_samples, dim):
    the data under study
gd: GridDescriptor instance:
density: array os shape(prod(gd.n_bins)):
    density of the model one the discrete grid implied by gd by default, this is recomputed

mpaxes: axes handle to make the figure, optional, :
    if None, a new figure is created

simple_update (x, z, plike)
This is a step in the sampling procedure that uses internal corss_validation

Parameters x: array of shape(n_samples, dim), :
    the input data
z: array of shape(n_samples), :
    the associated membership variables
plike: array of shape(n_samples), :
the likelihood under the prior

Returns  like: array od shape(n_samples),  :

the likelihood of the data

test (x, tiny=1e-15)
Returns the log-likelihood of the mixture for x

Parameters  x array of shape (n_samples,self.dim) :

the data used in the estimation process

Returns  ll: array of shape(n_samples) :

the log-likelihood of the rows of x

train (x, z=None
Idem initialize_and_estimate

unweighted_likelihood (x)
return the likelihood of each data for each component the values are not weighted by the component weights

Parameters  x: array of shape (n_samples,self.dim) :

the data used in the estimation process

Returns  like, array of shape(n_samples,self.k) :

unweighted component-wise likelihood

Notes

Hopefully faster

unweighted_likelihood_ (x)
return the likelihood of each data for each component the values are not weighted by the component weights

Parameters  x: array of shape (n_samples,self.dim) :

the data used in the estimation process

Returns  like, array of shape(n_samples,self.k) :

unweighted component-wise likelihood

update (x, z)
Update function (draw a sample of the IMM parameters)

Parameters  x array of shape (n_samples,self.dim) :

the data used in the estimation process

z array of shape (n_samples), type = np.int :

the corresponding classification

update_means (x, z)
Given the allocation vector z, and the corresponding data x, resample the mean

Parameters  x: array of shape (nb_samples,self.dim) :

the data used in the estimation process
**z**: array of shape (nb_samples), type = np.int :

the corresponding classification

**update_precisions** (x, z)

Given the allocation vector z, and the corresponding data x, resample the precisions

**Parameters**

- **x** array of shape (nb_samples, self.dim) :
  
  the data used in the estimation process

- **z** array of shape (nb_samples), type = np.int :
  
  the corresponding classification

**update_weights** (z)

Given the allocation vector z, resample the weights parameter

**Parameters**

- **z** array of shape (n_samples), type = np.int :
  
  the allocation variable

### 24.2.2 MixedIMM

class nipy.algorithms.clustering.imm.MixedIMM(alpha=0.5, dim=1)

Bases: nipy.algorithms.clustering.imm.IMM

Particular IMM with an additional null class. The data is supplied together with a sample-related probability of being under the null.

**Methods**

- `average_log_like(x[, tiny])` returns the averaged log-likelihood of the mode for the dataset x

- `bayes_factor(x, z[, nperm, verbose])` Evaluate the Bayes Factor of the current model using Chib's method

- `bic(like[, tiny])` Computation of bic approximation of evidence

- `check()` Checking the shape of different matrices involved in the model

- `check_x(x)` essentially check that x.shape[1]==self.dim

- `conditional_posterior_proba(x, z[, perm])` Compute the probability of the current parameters of self

- `cross_validated_update(x, z, plike, ...[, kfold])` This is a step in the sampling procedure

- `estimate(x[, niter, delta, verbose])` Estimation of the model given a dataset x

- `evidence(x, z[, nperm, verbose])` See bayes_factor(self, x, z, nperm=0, verbose=0)

- `guess_priors(x[, nocheck])` Set the priors in order of having them weakly uninformative

- `guess_regularizing(x[, bcheck])` Set the regularizing priors as weakly informative

- `initialize(x)` initialize z using a k-means algorithm, then update the parameters

- `initialize_and_estimate(x[, z, niter, ...])` Estimation of self given x

- `likelihood(x[, plike])` return the likelihood of the model for the data x

- `likelihood_under_the_prior(x)` Computes the likelihood of x under the prior

- `map_label(x[, like])` return the MAP labelling of x

- `mixture_likelihood(x)` Returns the likelihood of the mixture for x

- `plugin(means, precisions, weights)` Set manually the weights, means and precision of the model

- `pop(z)` compute the population, i.e. the statistics of allocation

- `probability_under_prior()` Compute the probability of the current parameters of self

- `reduce(z)` Reduce the assignments by removing empty clusters and update self.k

- `sample(x, null_class_proba[, niter, ...])` sample the indicator and parameters

- `sample_and_average(x[, niter, verbose])` sample the indicator and parameters

Continued on next page
Table 24.2 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>sample_indicator(like, null_class_proba)</code></td>
<td>Sample the indicator from the likelihood</td>
</tr>
<tr>
<td><code>set_constant_densities([null_dens, prior_dens])</code></td>
<td>Set the null and prior densities as constant</td>
</tr>
<tr>
<td><code>set_priors(x)</code></td>
<td>Set the priors in order of having them weakly uninformative</td>
</tr>
<tr>
<td><code>show(x, gd[, density, axes])</code></td>
<td>Function to plot a GMM, still in progress</td>
</tr>
<tr>
<td><code>show_components(x, gd[, density, mpaxes])</code></td>
<td>Function to plot a GMM – Currently, works only in 1D</td>
</tr>
<tr>
<td><code>simple_update(x, z, plike, null_class_proba)</code></td>
<td>One step in the sampling procedure (one data sweep)</td>
</tr>
<tr>
<td><code>test(x[, tiny])</code></td>
<td>Returns the log-likelihood of the mixture for x</td>
</tr>
<tr>
<td><code>train(x[, z, niter, delta, ninit, verbose])</code></td>
<td>Idem initialize_and_estimate</td>
</tr>
<tr>
<td><code>unweighted_likelihood(x)</code></td>
<td>Return the likelihood of each data for each component</td>
</tr>
<tr>
<td><code>unweighted_likelihood_(x)</code></td>
<td>Return the likelihood of each data for each component</td>
</tr>
<tr>
<td><code>update(x, z)</code></td>
<td>Update function (draw a sample of the IMM parameters)</td>
</tr>
<tr>
<td><code>update_means(x, z)</code></td>
<td>Given the allocation vector z,</td>
</tr>
<tr>
<td><code>update_precisions(x, z)</code></td>
<td>Given the allocation vector z,</td>
</tr>
<tr>
<td><code>update_weights(z)</code></td>
<td>Given the allocation vector z, resample the weights parameter</td>
</tr>
</tbody>
</table>

`__init__(alpha=0.5, dim=1)`

Parameters:
- `alpha`: float, optional, the parameter for cluster creation
- `dim`: int, optional, the dimension of the the data

Note: use the function `set_priors()` to set adapted priors:

`average_log_like(x, tiny=1e-15)`

Returns the averaged log-likelihood of the mode for the dataset x

Parameters:
- `x`: array of shape (n_samples,self.dim), the data used in the estimation process
- `tiny = 1.e-15`: a small constant to avoid numerical singularities

`bayes_factor(x, z, nperm=0, verbose=0)`

Evaluate the Bayes Factor of the current model using Chib’s method

Parameters:
- `x`: array of shape (nb_samples,dim), the data from which bic is computed
- `z`: array of shape (nb_samples), type = np.int, the corresponding classification
- `nperm=0`: int, the number of permutations to sample to model the label switching issue in the computation of the Bayes Factor
- `verbose=0`: verbosity mode

Returns `bf (float)` the computed evidence (Bayes factor)

Notes

\texttt{bic(like, tiny=1e-15)}

Computation of bic approximation of evidence

**Parameters**

- \texttt{like}, array of shape (n\_samples, self.k):
  component-wise likelihood

- \texttt{tiny=1.e-15}, a small constant to avoid numerical singularities:

**Returns**

- the bic value, float:

\texttt{check()}

Checking the shape of different matrices involved in the model

\texttt{check\_x(x)}

essentially check that x.shape[1]==self.dim

x is returned with possibly reshaping

\texttt{conditional\_posterior\_proba(x, z, perm=None)}

Compute the probability of the current parameters of self given x and z

**Parameters**

- \texttt{x: array of shape (nb\_samples, dim)},:
  the data from which bic is computed

- \texttt{z: array of shape (nb\_samples), type = np.int,}:
  the corresponding classification

- \texttt{perm: array of shape(nperm, self.k), typ=np.int, optional}:
  all permutation of z under which things will be recomputed By default, no permutation is performed

\texttt{cross\_validated\_update(x, z, plike, null\_class\_proba, kfold=10)}

This is a step in the sampling procedure that uses internal cross-validation

**Parameters**

- \texttt{x: array of shape(n\_samples, dim)},:
  the input data

- \texttt{z: array of shape(n\_samples),}:
  the associated membership variables

- \texttt{plike: array of shape(n\_samples),}:
  the likelihood under the prior

- \texttt{kfold: int, optional, or array}:
  number of folds in cross-validation loop or set of indexes for the cross-validation procedure

- \texttt{null\_class\_proba: array of shape(n\_samples),}:
  prior probability to be under the null

**Returns**

- \texttt{like: array of shape(n\_samples),}:
  the (cross-validated) likelihood of the data

- \texttt{z: array of shape(n\_samples),}:
  the associated membership variables
Notes

When kfold is an array, there is an internal reshuffling to randomize the order of updates

**estimate** \((x, \text{niter}=100, \text{delta}=0.0001, \text{verbose}=0)\)

Estimation of the model given a dataset \(x\)

**Parameters**

- \(x\) array of shape \((n\_samples, \text{dim})\):
  - the data from which the model is estimated
- \(\text{niter}=100\): maximal number of iterations in the estimation process:
- \(\text{delta} = 1.e-4\): increment of data likelihood at which convergence is declared
- \(\text{verbose}=0\): verbosity mode:

**Returns**

- \(\text{bic}\): an asymptotic approximation of model evidence

**evidence** \((x, z, nperm=0, \text{verbose}=0)\)

See bayes_factor(self, x, z, nperm=0, verbose=0)

**guess_priors** \((x, \text{nocheck}=0)\)

Set the priors in order of having them weakly uninformative this is from Fraley and raftery; Journal of Classification 24:155-181 (2007)

**Parameters**

- \(x\), array of shape \((nb\_samples, self\_dim)\):
  - the data used in the estimation process
- \(\text{nocheck}\): boolean, optional:
  - if nocheck==True, check is skipped

**guess_regularizing** \((x, bcheck=1)\)

Set the regularizing priors as weakly informative according to Fraley and raftery; Journal of Classification 24:155-181 (2007)

**Parameters**

- \(x\) array of shape \((n\_samples, \text{dim})\):
  - the data used in the estimation process

**initialize** \((x)\)

initialize \(z\) using a k-means algorithm, then update the parameters

**Parameters**

- \(x\): array of shape \((nb\_samples, \text{self}\_dim)\):
  - the data used in the estimation process

**initialize_and_estimate** \((x, z=\text{None}, \text{niter}=100, \text{delta}=0.0001, \text{ninit}=1, \text{verbose}=0)\)

Estimation of self given \(x\)

**Parameters**

- \(x\) array of shape \((n\_samples, \text{dim})\):
  - the data from which the model is estimated
- \(z\) = \text{None}: array of shape \((n\_samples)\):
  - a prior labelling of the data to initialize the computation
- \(\text{niter}=100\): maximal number of iterations in the estimation process:
- \(\text{delta} = 1.e-4\): increment of data likelihood at which:
  - convergence is declared
ninit=1: number of initialization performed:

to reach a good solution

verbose=0: verbosity mode:

Returns  the best model is returned:

likelihood(x, plike=None)
return the likelihood of the model for the data x the values are weighted by the components weights

Parameters  x: array of shape (n_samples, self.dim),:
the data used in the estimation process
plike: array os shape (n_samples), optional,x :
the desnity of each point under the prior

Returns  like, array of shape(nbitem,self.k) :
component-wise likelihood :

likelihood_under_the_prior(x)
Computes the likelihood of x under the prior

Parameters  x, array of shape (self.n_samples,self.dim) :

Returns  w, the likelihood of x under the prior model (unweighted) :

map_label(x, like=None)
return the MAP labelling of x

Parameters  x array of shape (n_samples,dim) :
the data under study
like=None array of shape(n_samples,self.k) :
component-wise likelihood if like==None, it is recomputed

Returns  z: array of shape(n_samples): the resulting MAP labelling :
of the rows of x

mixture_likelihood(x)
Returns the likelihood of the mixture for x

Parameters  x: array of shape (n_samples,self.dim) :
the data used in the estimation process

plugin(means, precisions, weights)
Set manually the weights, means and precision of the model

Parameters  means: array of shape (self.k,self.dim) :
precisions: array of shape (self.k,self.dim,self.dim) :
or (self.k, self.dim)
weights: array of shape (self.k) :

pop(z)
compute the population, i.e. the statistics of allocation

Parameters  z array of shape (nb_samples), type = np.int :
the allocation variable
Returns `hist` : array shape (self.k) count variable

```python
probability_under_prior()
```
Compute the probability of the current parameters of self given the priors

```python
reduce(z)
```
Reduce the assignments by removing empty clusters and update self.k

Parameters `z` : array of shape(n), :

- a vector of membership variables changed in place

Returns `z` : the remapped values :

```python
sample(x, null_class_proba, niter=1, sampling_points=None, init=False, kfold=None, co_clustering=False, verbose=0)
```
sample the indicator and parameters

Parameters `x` : array of shape (n_samples, self.dim), :

- the data used in the estimation process

`null_class_proba` : array of shape(n_samples), :

- the probability to be under the null

`niter` : int, :

- the number of iterations to perform

`sampling_points` : array of shape(nbpoints, self.dim), optional :

- points where the likelihood will be sampled this defaults to x

`kfold` : int, optional, :

- parameter of cross-validation control by default, no cross-validation is used the procedure is faster but less accurate

`co_clustering` : bool, optional :

- if True, return a model of data co-labelling across iterations

`verbose=0` : verbosity mode :

Returns `likelihood` : array of shape(nbpoints) :

- total likelihood of the model

`pproba` : array of shape(n_samples), :

- the posterior of being in the null (the posterior of null_class_proba)

`coclust` : only if co_clustering==True, :

- sparse_matrix of shape (n_samples, n_samples), frequency of co-labelling of each sample pairs across iterations

```python
sample_and_average(x, niter=1, verbose=0)
```
sample the indicator and parameters the average values for weights,means, precisions are returned

Parameters `x` = array of shape (nb_samples,dim) :

- the data from which bic is computed

`niter=1` : number of iterations :
Returns  weights: array of shape (self.k) :
  means: array of shape (self.k, self.dim) :
  precisions: array of shape (self.k, self.dim, self.dim) :
    or (self.k, self.dim) these are the average parameters across samplings

Notes

All this makes sense only if no label switching as occurred so this is wrong in general (asymptotically).
fix: implement a permutation procedure for components identification

sample_indicator (like, null_class_proba)  
sample the indicator from the likelihood

Parameters  like: array of shape (nbitem, self.k) :
  component-wise likelihood

null_class_proba: array of shape(n_samples), :
  prior probability to be under the null

Returns  z: array of shape(nbitem): a draw of the membership variable :

Notes

Here z=-1 encodes for the null class

set_constant_densities (null_dens=None, prior_dens=None)  
Set the null and prior densities as constant (over a supposedly compact domain)

Parameters  null_dens: float, optional :
  constant for the null density

prior_dens: float, optional :
  constant for the prior density

set_priors (x)  
Set the priors in order of having them weakly uninformative this is from Fraley and raftery; Journal of Classification 24:155-181 (2007)

Parameters  x, array of shape (n_samples, self.dim) :
  the data used in the estimation process

show (x, gd, density=None, axes=None)  
Function to plot a GMM, still in progress Currently, works only in 1D and 2D

Parameters  x: array of shape(n_samples, dim) :
  the data under study

gd: GridDescriptor instance :

density: array os shape(prod(gd.n_bins)) :
  density of the model one the discrete grid implied by gd by default, this is recomputed

show_components (x, gd, density=None, mpaxes=None)  
Function to plot a GMM – Currently, works only in 1D
Parameters  
x: array of shape(n_samples, dim) :
  the data under study

gd: GridDescriptor instance :

density: array os shape(prod(gd.n_bins)) :
  density of the model one the discrete grid implied by gd by default, this is recomputed

mpaxes: axes handle to make the figure, optional, :
  if None, a new figure is created

**simple_update**(x, z, plike, null_class_proba)
  One step in the sampling procedure (one data sweep)

Parameters  
x: array of shape(n_samples, dim), :
  the input data

z: array of shape(n_samples), :
  the associated membership variables

plike: array of shape(n_samples), :
  the likelihood under the prior

null_class_proba: array of shape(n_samples), :
  prior probability to be under the null

Returns  
like: array od shape(n_samples), :
  the likelihood of the data under the H1 hypothesis

**test**(x, tiny=1e-15)
  Returns the log-likelihood of the mixture for x

Parameters  
x array of shape (n_samples,self.dim) :
  the data used in the estimation process

Returns  
ll: array of shape(n_samples) :
  the log-likelihood of the rows of x

**train**(x, z=None, niter=100, delta=0.0001, ninit=1, verbose=0)
  Idem initialize_and_estimate

unweighted_likelihood(x)
  return the likelihood of each data for each component the values are not weighted by the component weights

Parameters  
x: array of shape (n_samples,self.dim) :
  the data used in the estimation process

Returns  
like, array of shape(n_samples,self.k) :
  unweighted component-wise likelihood
Notes

Hopefully faster

\texttt{unweighted\_likelihood}(x)

return the likelihood of each data for each component the values are not weighted by the component weights

\textbf{Parameters}  
\texttt{x}: array of shape (n\_samples, self\_dim):
the data used in the estimation process

\textbf{Returns}  
\texttt{like}, array of shape(n\_samples, self\_k):
unweighted component-wise likelihood

\texttt{update}(x, z)

Update function (draw a sample of the IMM parameters)

\textbf{Parameters}  
\texttt{x} array of shape (n\_samples, self\_dim):
the data used in the estimation process

\texttt{z} array of shape (n\_samples), type = np\_int:
the corresponding classification

\texttt{update\_means}(x, z)

Given the allocation vector \( z \), and the corresponding data \( x \), resample the mean

\textbf{Parameters}  
\texttt{x}: array of shape (nb\_samples, self\_dim):
the data used in the estimation process

\texttt{z}: array of shape (nb\_samples), type = np\_int:
the corresponding classification

\texttt{update\_precisions}(x, z)

Given the allocation vector \( z \), and the corresponding data \( x \), resample the precisions

\textbf{Parameters}  
\texttt{x} array of shape (nb\_samples, self\_dim):
the data used in the estimation process

\texttt{z} array of shape (nb\_samples), type = np\_int:
the corresponding classification

\texttt{update\_weights}(z)

Given the allocation vector \( z \), resmaple the weights parameter

\textbf{Parameters}  
\texttt{z} array of shape (n\_samples), type = np\_int:
the allocation variable

\section{24.3 Functions}

\texttt{nipy.algorithms.clustering.imm.co\_labelling}(z, kmax=None, kmin=None)

return a sparse co-labelling matrix given the label vector \( z \)

\textbf{Parameters}  
\texttt{z}: array of shape(n\_samples), :
the input labels
kmax: int, optional, :
    considers only the labels in the range [0, kmax]

Returns  colabel: a sparse coo_matrix, :
    yields the co labelling of the data i.e. c[i,j]= 1 if z[i]==z[j], 0 otherwise

nipy.algorithms.clustering.imm.main()
Illustrative example of the behaviour of imm
ALGORITHMS.CLUSTERING_Utils

25.1 Module: algorithms.clustering.utils

25.2 Functions

nipy.algorithms.clustering.utils.kmeans(X, nbclusters=2, Labels=None, maxiter=300, delta=0.0001, verbose=0, ninit=1)

kmeans clustering algorithm

Parameters

- X: array of shape (n,p): n = number of items, p = dimension : data array
- nbclusters (int), the number of desired clusters :
- Labels = None array of shape (n) prior Labels. :
  if None or inadequate a random initialization is performed.
- maxiter=300 (int), the maximum number of iterations before convergence :
- delta: float, optional, : the relative increment in the results before declaring convergence.
- verbose: verbosity mode, optional :
- ninit: int, optional, number of random initializations :

Returns

- Centers: array of shape (nbclusters, p), : the centroids of the resulting clusters
- Labels : array of size n, the discrete labels of the input items
- J (float): the final value of the inertia criterion :

nipy.algorithms.clustering.utils.voronoï(x, centers)

Assignment of data items to nearest cluster center

Parameters

- x array of shape (n,p) :
  n = number of items, p = data dimension
- centers, array of shape (k, p) the cluster centers :

Returns

- z vector of shape(n), the resulting assignment :
26.1 Module: algorithms.clustering.von_mises_fisher_mixture

Inheritance diagram for nipy.algorithms.clustering.von_mises_fisher_mixture:

```
clustering.von_mises_fisher_mixture.VonMisesMixture
```

Implementation of Von-Mises-Fisher Mixture models, i.e. the equivalent of mixture of Gaussian on the sphere.

Author: Bertrand Thirion, 2010-2011

26.2 Class

26.3 VonMisesMixture

```
class nipy.algorithms.clustering.von_mises_fisher_mixture.VonMisesMixture(k, 
precision, 
means=None, 
weights=None, 
null_class=False)
```

Bases: object

Model for Von Mises mixture distribution with fixed variance on a two-dimensional sphere

Methods

```
density_per_component(x)  Compute the per-component density of the data
```

Continued on next page
Table 26.1 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>estimate(x[, maxiter, miniter, bias])</code></td>
<td>Return average log density across samples</td>
</tr>
<tr>
<td><code>estimate_means(x, z)</code></td>
<td>Calculate and set means from x and z</td>
</tr>
<tr>
<td><code>estimate_weights(z)</code></td>
<td>Calculate and set weights from z</td>
</tr>
<tr>
<td><code>log_density_per_component(x)</code></td>
<td>Compute the per-component density of the data</td>
</tr>
<tr>
<td><code>log_weighted_density(x)</code></td>
<td>Return log weighted density</td>
</tr>
<tr>
<td><code>mixture_density(x)</code></td>
<td>Return mixture density</td>
</tr>
<tr>
<td><code>responsibilities(x)</code></td>
<td>Return responsibilities</td>
</tr>
<tr>
<td><code>show(x)</code></td>
<td>Visualization utility</td>
</tr>
<tr>
<td><code>weighted_density(x)</code></td>
<td>Return weighted density</td>
</tr>
</tbody>
</table>

`__init__(k, precision, means=None, weights=None, null_class=False)`
Initialize Von Mises mixture

**Parameters**
- **k**: int, number of components
- **precision**: float, the fixed precision parameter
- **means**: array of shape(self.k, 3), optional, input component centers
- **weights**: array of shape(self.k), optional, input components weights
- **null_class**: bool, optional, Inclusion of a null class within the model (related to k=0)

`density_per_component(x)`
Compute the per-component density of the data

**Parameters**
- **x**: array of shape(n,3), should be on the unit sphere

**Returns**
- **like**: array of shape(n, self.k), with non-negative values, the density

`estimate(x, maxiter=100, miniter=1, bias=None)`
Return average log density across samples

**Parameters**
- **x**: array of shape(n,3), should be on the unit sphere
  - **maxiter**: int, optional, maximum number of iterations of the algorithms
  - **miniter**: int, optional, minimum number of iterations
  - **bias**: array of shape(n), optional, prior probability of being in a non-null class

**Returns**
- **ll**: float
average (across samples) log-density

**estimate_means**(x, z)
Calculate and set means from x and z

  Parameters  
  x: array of shape(n,3):
  should be on the unit sphere

  z: array of shape(self.k):

**estimate_weights**(z)
Calculate and set weights from z

  Parameters  
  z: array of shape(self.k):

**log_density_per_component**(x)
Compute the per-component density of the data

  Parameters  
  x: array of shape(n,3):
  should be on the unit sphere

  Returns  
  like: array of shape(n, self.k), with non-negative values:
  the density

**log_weighted_density**(x)
Return log weighted density

  Parameters  
  x: array of shape(n,3):
  should be on the unit sphere

  Returns  
  log_like: array of shape(n, self.k):

**mixture_density**(x)
Return mixture density

  Parameters  
  x: array of shape(n,3):
  should be on the unit sphere

  Returns  
  like: array of shape(n):

**responsibilities**(x)
Return responsibilities

  Parameters  
  x: array of shape(n,3):
  should be on the unit sphere

  Returns  
  resp: array of shape(n, self.k):

**show**(x)
Visualization utility

  Parameters  
  x: array of shape(n,3):
  should be on the unit sphere

**weighted_density**(x)
Return weighted density

  Parameters  
  x: array of shape(n,3):
  should be on the unit sphere

  Returns  
  like: array:
of shape(n, self.k)

### 26.4 Functions

```python
nipy.algorithms.clustering.von_mises_fisher_mixture.estimate_robust_vmm(k, precision, null_class, x, ninit=10, bias=None, maxiter=100)
```

Return the best von_mises mixture after several initialization

**Parameters**
- **k**: int, number of classes
  - precision: float, priori precision parameter
  - null_class: bool, optional
  - should a null class be included or not
- **x**: array fo shape(n,3)
  - input data, should be on the unit sphere
- **ninit**: int, optional
  - number of iterations
- **bias**: array of shape(n), optional
  - prior probability of being in a non-null class
- **maxiter**: int, optional
  - maximum number of iterations after each initialization

```python
nipy.algorithms.clustering.von_mises_fisher_mixture.example_cv_nonoise()

nipy.algorithms.clustering.von_mises_fisher_mixture.example_noisy()

nipy.algorithms.clustering.von_mises_fisher_mixture.select_vmm(krange, precision, null_class, x, ninit=10, bias=None, maxiter=100, verbose=0)
```

Return the best von_mises mixture after several initialization

**Parameters**
- **krange**: list of ints, 
  - number of classes to consider
- **precision**: 
- **null_class**: 
- **x**: array fo shape(n,3)
  - should be on the unit sphere
ninit: int, optional, :
    number of iterations
maxiter: int, optional, :
bias: array of shape(n), :
    a prior probability of not being in the null class
verbose: Bool, optional :

nipy.algorithms.clustering.von_mises_fisher_mixture.select_vmm_cv(krange, precision, x, null_class, cv_index, ninit=5, maxiter=100, bias=None, verbose=0)

Return the best von_mises mixture after severla initialization

Parameters  krange: list of ints, :
    number of classes to consider
precision: float, :
    precision parameter of the von-mises densities
x: array fo shape(n, 3) :
    should be on the unit sphere
null_class: bool, whether a null class should be included or not :
cv_index: set of indices for cross validation :
ninit: int, optional, :
    number of iterations
maxiter: int, optional, :
bias: array of shape (n), prior :

nipy.algorithms.clustering.von_mises_fisher_mixture.sphere_density(npoints)

Return the points and area of a npoints**2 points sampled on a sphere

Returns  s : array of shape(npoints ** 2, 3)
    area: array of shape(npoints) :
CHAPTER TWENTYSEVEN

ALGORITHMS.DIAGNOSTICS.SCREENS

27.1 Module: algorithms.diagnostics.screens

Diagnostic 4d image screen

27.2 Functions

nipy.algorithms.diagnostics.screens.screen(img4d, ncomp=10)

Diagnostic screen for 4d FMRI image

Includes PCA, tsdiffana and mean, std, min, max images.

Parameters

img4d : Image

4d image file

ncomp : int, optional

number of component images to return. Default is 10

Returns

screen : dict

with keys:

• mean : mean image (all summaries are over last dimension)
• std : standard deviation image
• max : image of max
• min : min
• pca : 4D image of PCA component images
• pca_res : dict of results from PCA
• ts_res : dict of results from tsdiffana

Examples

>>> import nipy as ni
>>> from nipy.testing import funcfile
>>> img = ni.load_image(funcfile)
>>> screen_res = screen(img)
>>> screen_res['mean'].ndim
3
>>> screen_res['pca'].ndim
4

```
nipy.algorithms.diagnostics.screens.write_screen_res(res, out_path, out_root,
out_img_ext='.nii',
pcnt_var_thresh=0.1)
```

Write results from `screen` to disk as images

**Parameters**

- `res`: dict
  - output from `screen` function
- `out_path`: str
  - directory to which to write output images
- `out_root`: str
  - part of filename between image-specific prefix and image-specific extension to use for writing images
- `out_img_ext`: str, optional
  - extension (identifying image type) to which to write volume images. Default is `.nii`
- `pcnt_var_thresh`: float, optional
  - threshold below which we do not plot percent variance explained by components; default is 0.1. This removes the long tail from percent variance plots.

**Returns**

- `None`:
28.1 Module: algorithms.diagnostics.timediff

Time series diagnostics

These started life as tsdiffana.m - see http://imaging.mrc-cbu.cam.ac.uk/imaging/DataDiagnostics

Oliver Josephs (FIL) gave me the idea of time-point to time-point subtraction as a diagnostic for motion and other sudden image changes.

nipy.algorithms.diagnostics.timediff.time_slice_diffs(arr,
            time_axis=-1,
            slice_axis=-2)

Time-point to time-point differences over volumes and slices

We think of the passed array as an image. The image has a “time” dimension given by time_axis and a “slice” dimension, given by slice_axis, and one or other dimensions. In the case of imaging there will usually be two more dimensions (the dimensions defining the size of an image slice). A single slice in the time dimension we call a “volume”. A single entry in arr is a “voxel”. For example, if time_axis == 0, then v = arr[0] would be the first volume in the series. The volume v above has v.size voxels. If, in addition, slice_axis == 1, then for the volume v (above) s = v[0] would be a “slice”, with s.size voxels. These are obviously terms from neuroimaging.

Parameters  

arr : array_like
    Array over which to calculate time and slice differences. We’ll call this array an ‘image’ in this doc.

time_axis : int
    axis of arr that varies over time.

slice_axis : int
    axis of arr that varies over image slice.

Returns  

results : dict
    Here T is the number of time points (arr.shape[time_axis]) and S is the number of slices (arr.shape[slice_axis]), v is the shape of a volume, and d2[t] is the volume of squared differences between voxels at time point t and time point t+1

results has keys:

- ‘volume_mean_diff2’ [(T-1,) array] array containing the mean (over voxels in volume) of the squared difference from one time point to the next

- ‘slice_mean_diff2’ [(T-1, S) array] giving the mean (over voxels in slice) of the difference from one time point to the next, one value per slice, per timepoint
• ‘volume_means’ [(T,) array] mean over voxels for each volume vol[t] for t in 0:T

• ‘slice_diff2_max_vol’ [v[: array] volume, of same shape as input volumes, where each slice is the slice from d2[t] for t in 0:T-1, that has the largest variance across t. Thus each slice in the volume may well result from a different difference time point.

• ‘diff2_mean_vol’ [v[:] array] volume with the mean of d2[t] across t for t in 0:T-1.
29.1 Module: algorithms.diagnostics.tsdiffplot

plot tsdiffana parameters

29.2 Functions

nipy.algorithms.diagnostics.tsdiffplot.plot_tsdiffs(results, axes=None)
Plotting routine for time series difference metrics
Requires matplotlib

Parameters results : dict
Results of format returned from nipy.algorithms.diagnostics.time_slice_diff()

nipy.algorithms.diagnostics.tsdiffplot.plot_tsdiffs_image(img, axes=None, show=True)
Plot time series diagnostics for image

Parameters img : image-like or filename str
image on which to do diagnostics
axes : None or sequence, optional
Aaxes on which to plot the diagnostics. If None, then we create a figure and subplots for
the plots. Sequence should have length >=4.
show : bool, optional
If True, show the figure after plotting it

Returns axes : Matplotlib axes
Axes on which we have done the plots. Will be same as axes input if not None
30.1 Module: algorithms.fwhm

Inheritance diagram for nipy.algorithms.fwhm:

```
algorithms.fwhm.Resels ---> algorithms.fwhm.ReselImage
```

This module provides classes and definitions for using full width at half maximum (FWHM) to be used in conjunction with Gaussian Random Field Theory to determine resolution elements (resels).

A resolution element (resel) is defined as a block of pixels of the same size as the FWHM of the smoothed image. There are two methods implemented to estimate (3d, or volumewise) FWHM based on a 4d Image:

- `fastFHWM`: used if the entire 4d Image is available
- `iterFWHM`: used when 4d Image is being filled in by slices of residuals

30.2 Classes

30.2.1 ReselImage
class nipy.algorithms.fwhm.ReselImage(resels=None, fwhm=None, **keywords)
    Bases: nipy.algorithms.fwhm.Resels

    Methods

    | Method            | Description                                      |
    |-------------------|--------------------------------------------------|
    | fwhm2resel(fwhm)  | Convert FWHM `fwhm` to equivalent resels per voxel |
    | integrate([mask]) | Integrate resels within `mask` (or use self.mask) |
    | resel2fwhm(resels)| Convert resels as `resels` to isotropic FWHM     |
__init__ (resels=None, fwhm=None, **keywords)
   Initialize resel image

   Parameters
       resels : core.api.Image
           Image of resel per voxel values.
       fwhm : core/api.Image
           Image of FWHM values.
       keywords : dict
           Passed as keywords arguments to core.api.Image

fwhm2resel (fwhm)
   Convert FWHM fwhm to equivalent resels per voxel

   Parameters
       fwhm : float
           Convert an FWHM value to an equivalent resels per voxel based on step sizes in self.coordmap.

   Returns
       resels : float

integrate (mask=None)
   Integrate resels within mask (or use self.mask)

   Parameters
       mask : Image
           Optional mask over which to integrate (add) resels.

   Returns
       total_resels : :
           the resels contained in the mask

       FWHM : float
           an estimate of FWHM based on the average resel per voxel

       nvoxel : int
           the number of voxels in the mask

resel2fwhm (resels)
   Convert resels as resels to isotropic FWHM

   Parameters
       resels : float
           Convert a resel value to an equivalent isotropic FWHM based on step sizes in self.coordmap.

   Returns
       fwhm : float

30.2.2 Resels

class nipy.algorithms.fwhm.Resels (
   coordmap, normalized=False, fwhm=None, resels=None,
   mask=None, clobber=False, D=3)
       Bases: object

       The Resels class.

       Methods
fwhm2resel(fwhm)  Convert FWHM \textit{fwhm} to equivalent reseels per voxel
integrate([mask]) Integrate resels within \textit{mask} (or use self.mask)
resel2fwhm(resels)  Convert resels as \textit{resels} to isotropic FWHM

__init__(\textit{coordmap}, \textit{normalized}=False, \textit{fwhm}=None, \textit{resels}=None, \textit{mask}=None, \textit{clobber}=False, \textit{D}=3)
Initialize resels class

Parameters  
\textit{coordmap} : CoordinateMap
CoordinateMap over which fwhm and resels are to be estimated. Used in fwhm/resel conversion.

\textit{fwhm} : Image
Optional Image of FWHM. Used to convert FWHM Image to resels if FWHM is not being estimated.

\textit{resels} : Image
Optional Image of resels. Used to compute resels within a mask, for instance, if FWHM has already been estimated.

\textit{mask} : Image
Mask over which to integrate resels.

\textit{clobber} : bool
Clobber output FWHM and resel images?

\textit{D} : int
Can be 2 or 3, the dimension of the final volume.

fwhm2resel(fwhm)  Convert FWHM \textit{fwhm} to equivalent reseels per voxel

Parameters  
\textit{fwhm} : float
Convert an FWHM value to an equivalent resels per voxel based on step sizes in self.coordmap.

Returns  
\textit{resels} : float

integrate([mask]=None) Integrate resels within \textit{mask} (or use self.mask)

Parameters  
\textit{mask} : Image
Optional mask over which to integrate (add) resels.

Returns  
\textit{total_resels} : 
the resels contained in the mask

\textit{FWHM} : float
an estimate of FWHM based on the average resel per voxel

\textit{nvoxel} : int : 
the number of voxels in the mask
resel2fwhm(resels)
Convert resels as resels to isotropic FWHM

Parameters
resels : float
Convert a resel value to an equivalent isotropic FWHM based on step sizes in self.coordmap.

Returns
fwhm : float
This module implements the BipartiteGraph class, used to represent weighted bipartite graph: it contains two types of vertices, say ‘left’ and ‘right’; then edges can only exist between ‘left’ and ‘right’ vertices. For simplicity the vertices of either side are labeled [1..V] and [1..W] respectively.

Author: Bertrand Thirion, 2006–2011

### 31.2 Class

### 31.3 BipartiteGraph

```python
class nipy.algorithms.graph.bipartite_graph.BipartiteGraph(V, W, edges=None, weights=None)
Bases: object

Bipartite graph class

A graph for which there are two types of nodes, such that edges can exist only between nodes of type 1 and type 2 (not within) fields of this class: V (int, > 0) the number of type 1 vertices W (int, > 0) the number of type 2 vertices E: (int) the number of edges edges: array of shape (self.E, 2) reprensenting pairwise neighbors weights, array of shape (self.E), +1/-1 for scending/descending links
```

Methods
copy() returns a copy of self

set_edges(edges) Set edges to graph sets self.edges=edges if 1.

set_weights(weights) Set weights weights to edges

subgraph_left(valid[, renumb]) Extraction of a subgraph

subgraph_right(valid[, renumb]) Extraction of a subgraph

__init__(V, W, edges=None, weights=None)
Constructor

Parameters
V (int), the number of vertices of subset 1:

W (int), the number of vertices of subset 2:

edges=None: array of shape (self.E, 2):
the edge array of the graph

weights=None: array of shape (self.E):
the asociated weights array

copy() returns a copy of self

set_edges(edges)
Set edges to graph

sets self.edges=edges if

1. edges has a correct size
2. edges take values in [0..V-1]*[0..W-1]

Parameters
edges: array of shape(self.E, 2): set of candidate edges:

set_weights(weights)
Set weights weights to edges

Parameters
weights, array of shape(self.V): edges weights:

subgraph_left(valid, renumb=True)
Extraction of a subgraph

Parameters
valid, boolean array of shape self.V:
renumb, boolean: renumbering of the (left) edges:

Returns
G : None or BipartiteGraph instance
A new BipartiteGraph instance with only the left vertices that are True. If
sum(valid)==0, None is returned

subgraph_right(valid, renumb=True)
Extraction of a subgraph

Parameters
valid : bool array of shape self.V
renumb : bool, optional
renumbering of the (right) edges

Returns
G : None or BipartiteGraph instance.
A new BipartiteGraph instance with only the right vertices that are True. If
sum(valid)==0, None is returned

31.4 Functions

nipy.algorithms.graph.bipartite_graph.bipartite_graph_from_adjacency(x)
  Instantiates a weighted graph from a square 2D array
  Parameters  x: 2D array instance, the input array :
  Returns  wg: BipartiteGraph instance :

nipy.algorithms.graph.bipartite_graph.bipartite_graph_from_coo_matrix(x)
  Instantiates a weighted graph from a (sparse) coo_matrix
  Parameters  x: scipy.sparse.coo_matrix instance, the input matrix :
  Returns  bg: BipartiteGraph instance :

nipy.algorithms.graph.bipartite_graph.check_feature_matrices(X, Y)
  checks whether the dimension of X and Y are consistent
  Parameters  X, Y arrays of shape (n1, p) and (n2, p) :
    where p = common dimension of the features :

nipy.algorithms.graph.bipartite_graph.cross_eps(X, Y, eps=1.0)
  Return the eps-neighbours graph of from X to Y
  Parameters  X, Y arrays of shape (n1, p) and (n2, p) :
    where p = common dimension of the features :
    eps=1, float: the neighbourhood size considered :
  Returns  the resulting bipartite graph instance :

Notes

for the sake of speed it is advisable to give PCA-preprocessed matrices X and Y.

nipy.algorithms.graph.bipartite_graph.cross_knn(X, Y, k=1)
  return the k-nearest-neighbours graph of from X to Y
  Parameters  X, Y arrays of shape (n1, p) and (n2, p) :
    where p = common dimension of the features :
    eps=1, float: the neighbourhood size considered :
  Returns  BipartiteGraph instance :

Notes

For the sake of speed it is advised to give PCA-transformed matrices X and Y.
32.1 Module: algorithms.graph.field

This module implements the Field class, which simply a WeightedGraph (see the graph.py) module, plus an array that yields (possibly multi-dimensional) features associated with graph vertices. This allows some kinds of computations (all those relating to mathematical morphology, diffusion etc.)

Certain functions are provided to Instantiate Fields easily, given a WeightedGraph and feature data.

Author: Bertrand Thirion, 2006–2011

32.2 Class

32.3 Field

```python
class nipy.algorithms.graph.field.Field(V, edges=None, weights=None, field=None):
```

Bases: nipy.algorithms.graph.graph.WeightedGraph

This is the basic field structure, which contains the weighted graph structure plus an array of data (the ‘field’)

t field is an array of size(n, p) where n is the number of vertices of the graph and p is the field dimension

Methods

- `adjacency()` returns the adjacency matrix of the graph as a sparse coo matrix
- `anti_symmeterize()` anti-symmeterize self, i.e. produces the graph
- `cc()` Compute the different connected components of the graph.
Table 32.1 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cliques()</td>
<td>Extraction of the graph cliques</td>
</tr>
<tr>
<td>closing([nbiter])</td>
<td>Morphological closing of the field data.</td>
</tr>
<tr>
<td>compact_neighbo()</td>
<td>Returns a compact representation of self</td>
</tr>
<tr>
<td>constrained_voronoise(seed)</td>
<td>Voronoï parcellation of the field starting from the input seed</td>
</tr>
<tr>
<td>copy()</td>
<td>Copy function</td>
</tr>
<tr>
<td>custom_watershed([refdim, th])</td>
<td>Customized watershed analysis of the field</td>
</tr>
<tr>
<td>cut_redundancies()</td>
<td>Returns a graph with redundant edges removed: each edge (ab) is present only once</td>
</tr>
<tr>
<td>degrees()</td>
<td>Returns the degree of the graph vertices.</td>
</tr>
<tr>
<td>diffusion([nbiter])</td>
<td>Diffusion of the field data in the weighted graph structure</td>
</tr>
<tr>
<td>dijkstra([seed])</td>
<td>Returns all the geodesic distances starting from seed</td>
</tr>
<tr>
<td>dilation([nbiter, fast])</td>
<td>Morphological dimlation of the field data.</td>
</tr>
<tr>
<td>erosion([nbiter])</td>
<td>Morphological opening of the field</td>
</tr>
<tr>
<td>floyd([seed])</td>
<td>Compute all the geodesic distances starting from seeds</td>
</tr>
<tr>
<td>from_3d_grid(xyz[, k])</td>
<td>Sets the graph to be the topological neighbours graph</td>
</tr>
<tr>
<td>geodesic_kmeans([seeds, label, maxiter, ...])</td>
<td>Geodesic k-means algorithm i.e.</td>
</tr>
<tr>
<td>get_E()</td>
<td>To get the number of edges in the graph</td>
</tr>
<tr>
<td>get_V()</td>
<td>To get the number of vertices in the graph</td>
</tr>
<tr>
<td>get_edges()</td>
<td>To get the graph’s edges</td>
</tr>
<tr>
<td>get_field()</td>
<td></td>
</tr>
<tr>
<td>get_local_maxima([refdim, th])</td>
<td>Look for the local maxima of one dimension (refdim) of self.field</td>
</tr>
<tr>
<td>get_vertices()</td>
<td>To get the graph’s vertices (as id)</td>
</tr>
<tr>
<td>get_weights()</td>
<td></td>
</tr>
<tr>
<td>highest_neighbor([refdim])</td>
<td>Computes the neighbor with highest field value along refdim</td>
</tr>
<tr>
<td>is_connected()</td>
<td>States whether self is connected or not</td>
</tr>
<tr>
<td>kruskal()</td>
<td>Creates the Minimum Spanning Tree of self using Kruskal’s algo.</td>
</tr>
<tr>
<td>left_incidence()</td>
<td>Return left incidence matrix</td>
</tr>
<tr>
<td>list_of_neighbors()</td>
<td>Returns the set of neighbors of self as a list of arrays</td>
</tr>
<tr>
<td>local_maxima([refdim, th])</td>
<td>Returns all the local maxima of a field</td>
</tr>
<tr>
<td>main_cc()</td>
<td>Returns the indexes of the vertices within the main cc</td>
</tr>
<tr>
<td>normalize([c])</td>
<td>Normalize the graph according to the index c</td>
</tr>
<tr>
<td>opening([nbiter])</td>
<td>Morphological opening of the field data.</td>
</tr>
<tr>
<td>remove_edges(valid)</td>
<td>Removes all the edges for which valid==0</td>
</tr>
<tr>
<td>remove_trivial_edges()</td>
<td>Removes trivial edges, i.e.</td>
</tr>
<tr>
<td>right_incidence()</td>
<td>Return right incidence matrix</td>
</tr>
<tr>
<td>set_edges(edges)</td>
<td>Sets the graph’s edges</td>
</tr>
<tr>
<td>set_euclidean(X)</td>
<td>Compute the weights of the graph as the distances between the</td>
</tr>
<tr>
<td>set_field(field)</td>
<td></td>
</tr>
<tr>
<td>set_gaussian(X[, sigma])</td>
<td>Compute the weights of the graph as a gaussian function</td>
</tr>
<tr>
<td>set_weights(weights)</td>
<td>Set edge weights</td>
</tr>
<tr>
<td>show([X, ax])</td>
<td>Plots the current graph in 2D</td>
</tr>
<tr>
<td>subfield(valid)</td>
<td>Returns a subfield of self, with only vertices such that valid &gt; 0</td>
</tr>
<tr>
<td>subgraph(valid)</td>
<td>Creates a subgraph with the vertices for which valid&gt;0</td>
</tr>
<tr>
<td>symmeterize()</td>
<td>Symmeterize self, modify edges and weights so that</td>
</tr>
<tr>
<td>threshold_bifurcations([refdim, th])</td>
<td>Analysis of the level sets of the field:</td>
</tr>
<tr>
<td>to_coo_matrix()</td>
<td>Return adjacency matrix as coo sparse</td>
</tr>
<tr>
<td>voronoï_diagram(seeds, samples)</td>
<td>Defines the graph as the Voronoï diagram (VD)</td>
</tr>
<tr>
<td>voronoï_labelling(seed)</td>
<td>Performs a voronoï labelling of the graph</td>
</tr>
<tr>
<td>ward(nbcluster)</td>
<td>Ward’s clustering of self</td>
</tr>
</tbody>
</table>

```
__init__(V, edges=None, weights=None, field=None)
```
Parameters
V (int > 0) the number of vertices of the graph:

edges=None: the edge array of the graph:

weights=None: the associated weights array:

field=None: the field data itself:

adjacency()
returns the adjacency matrix of the graph as a sparse coo matrix

Returns adj: scipy.sparse matrix instance, that encodes the adjacency matrix of self

anti_symmeterize()
anti-symmeterize self, i.e. produces the graph whose adjacency matrix would be the antisymmetric part of its current adjacency matrix

cc()
Compute the different connected components of the graph.

Returns label: array of shape(self.V), labelling of the vertices:

cliques()
Extraction of the graph's cliques these are defined using replicator dynamics equations

Returns cliques: array of shape (self.V), type (np.int):
labelling of the vertices according to the clique they belong to

closing(nbiter=1)
Morphological closing of the field data. self.field is changed inplace

Parameters nbiter=1: the number of iterations required

compact_neighb()
returns a compact representation of self

Returns idx: array of of shape(self.V + 1):
the positions where to find the neighbors of each node within neighb and weights

neighb: array of shape(self.E), concatenated list of neighbors:
weights: array of shape(self.E), concatenated list of weights:

constrained_voronoi(seed)
Voronoi parcellation of the field starting from the input seed

Parameters seed: int array of shape(p), the input seeds:

Returns label: The resulting labelling of the data:

copy()
copy function

custom_watershed(refdim=0, th=-inf)
customized watershed analysis of the field. Note that basins are found around each maximum (and not minimum as conventionally)

Parameters refdim: int, optional:
th: float optional, threshold of the field:

Returns idx: array of shape (nbassins):
indices of the vertices that are local maxima

32.3. Field
Neuroimaging in Python Documentation, Release 0.3.0

label : array of shape (self.V)
labelling of the vertices according to their bassin

cut_redundancies()
Returns a graph with redundant edges removed: each edge (ab) is present only once in the edge matrix: the corresponding weights are added.

Counts the resulting WeightedGraph:

degrees()
Returns the degree of the graph vertices.

Returns:
rdegree: (array, type=int, shape=(self.V,)), the right degrees:
ldegree: (array, type=int, shape=(self.V,)), the left degrees:

diffusion(nbiter=1)
diffusion of the field data in the weighted graph structure self.field is changed inplace

Parameters:
nbiter: int, optional the number of iterations required:

Notes
The process is run for all the dimensions of the field

dijkstra (seed=0)
Returns all the [graph] geodesic distances starting from seed x Parameters:

seed (int, >=1, <self.V)
or array of shape(p)
edge(s) from which the distances are computed

Returns:
dg: array of shape (self.V),
the graph distance dg from any vertex to the nearest seed

Notes
It is mandatory that the graph weights are non-negative

dilation (nbiter=1, fast=True)
Morphological dilation of the field data. self.field is changed

Parameters:
nbiter: int, optional, the number of iterations required:

erosion (nbiter=1)
Morphological opening of the field

Parameters:
nbiter: int, optional, the number of iterations required:

floyd (seed=None)
Compute all the geodesic distances starting from seeds

Parameters:
seed=None: array of shape (nbseed), type np.int:
vertex indexes from which the distances are computed if seed==None, then every edge
is a seed point

Returns:
dg array of shape (nbseed, self.V),
the graph distance dg from each seed to any vertex
Notes

It is mandatory that the graph weights are non-negative. The algorithm proceeds by repeating Dijkstra’s algo for each seed. Floyd’s algo is not used (O(self.V)^3 complexity...)

\texttt{from}_\texttt{3d\_grid}(\texttt{xyz, k=18})

Sets the graph to be the topological neighbours graph of the three-dimensional coordinates set xyz, in the k-connectivity scheme

Parameters \texttt{xyz: array of shape (self.V, 3) and type np.int, :}

\texttt{k = 18: the number of neighbours considered. (6, 18 or 26)}

Returns \texttt{E(int): the number of edges of self :}

\texttt{geodesic\_kmeans}(\texttt{seeds=None, label=None, maxiter=100, eps=0.0001, verbose=0})

Geodesic k-means algorithm i.e. obtention of clusters that are topologically connected and minimally variable concerning the information of self.field

Parameters \texttt{seeds: array of shape(p), optional, :}

initial indices of the seeds within the field if seeds==None the labels are used as initialization

\texttt{labels: array of shape(self.V) initial labels, optional, :}

it is expected that labels take their values in a certain range (0..lmax) if Labels==None, this is not used if seeds==None and labels==None, an exception is raised

\texttt{maxiter: int, optional, :}

maximal number of iterations

\texttt{eps: float, optional, :}

increase of inertia at which convergence is declared

Returns \texttt{seeds: array of shape (p), the final seeds :}

\texttt{label : array of shape (self.V), the resulting field label}

\texttt{J: float, inertia value :}

\texttt{get\_E()}

To get the number of edges in the graph

\texttt{get\_V()}

To get the number of vertices in the graph

\texttt{get\_edges()}

To get the graph’s edges

\texttt{get\_field()}

\texttt{get\_local\_maxima(refdim=0, th=-inf)}

Look for the local maxima of one dimension (refdim) of self.field

Parameters \texttt{refdim (int) the field dimension over which the maxima are looked after :}

\texttt{th = float, optional :}

threshold so that only values above th are considered

Returns \texttt{idx: array of shape (nmax) :}

indices of the vertices that are local maxima
depth: array of shape (nmax):

topological depth of the local maxima: depth[\text{idx}[i]] = q means that \text{idx}[i] is a q-order maximum

\text{get_vertices}()
To get the graph’s vertices (as id)

\text{get_weights}()

\text{highest_neighbor}(refdim=0)
Comes the neighbor with highest field value along refdim

Parameters refdim: int optional, the dimension to consider:

Returns hneighb: array of shape(self.V), index of the neighbor with highest value

\text{is_connected}()
States whether self is connected or not

\text{kruskal}()
Creates the Minimum Spanning Tree of self using Kruskal’s algo. efficient is self is sparse

Returns K, WeightedGraph instance: the resulting MST:

Notes
If self contains several connected components, will have the same number k of connected components

\text{left_incidence}()
Return left incidence matrix

Returns left_incid: list:
the left incidence matrix of self as a list of lists: i.e. the list[[e.0.0, ..., e.0.i(0)], ..., [e.V, E.V.i(V)]] where e.i.j is the set of edge indexes so that e.i.j[0] = i

\text{list_of_neighbors}()
returns the set of neighbors of self as a list of arrays

\text{local_maxima}(refdim=0, th=-inf)
Returns all the local maxima of a field

Parameters refdim (int) field dimension over which the maxima are looked after:

th: float, optional:
threshold so that only values above th are considered

Returns depth: array of shape (nmax):

a labelling of the vertices such that depth[v] = 0 if v is not a local maximum depth[v] = 1 if v is a first order maximum... depth[v] = q if v is a q-order maximum

\text{main_cc}()
Returns the indexes of the vertices within the main cc

Returns idx: array of shape (sizeof main cc):

\text{normalize}(c=0)
Normalize the graph according to the index c Normalization means that the sum of the edges values that go into or out each vertex must sum to 1

Parameters  

c=0 in \{0, 1, 2\}, optional: index that designates the way:

according to which D is normalized:

c == 0 => for each vertex a, sum{edge[e, 0]=a} D[e]=1
c == 1 => for each vertex b, sum{edge[e, 1]=b} D[e]=1
c == 2 => symmetric (‘l2’) normalization

Notes

Note that when sum_{edge[e, .] == a } D[e] = 0, nothing is performed

opening (nbiter=1)

Morphological opening of the field data. self.field is changed inplace

Parameters  

nbiter: int, optional, the number of iterations required:

remove_edges (valid)

Removes all the edges for which valid==0

Parameters  

valid : (self.E,) array

remove_trivial_edges ()

Removes trivial edges, i.e. edges that are (vv)-like. self.weights and self.E are corrected accordingly

Returns  

self.E (int): The number of edges:

right_incidence ()

Return right incidence matrix

Returns  

right_incid: list:

the right incidence matrix of self as a list of lists: i.e. the list[[e.0.0, ..., e.0.i(0)], ..., [e.V.0, E.V.i(V)]] where e.i.j is the set of edge indexes so that e.i.j[1] = i

set_edges (edges)

Sets the graph’s edges

set_euclidian (X)

Compute the weights of the graph as the distances between the corresponding rows of X, which represents an embedding of self

Parameters  

X array of shape (self.V, edim),:

the coordinate matrix of the embedding

set_field (field)

set_gaussian (X, sigma=0)

Compute the weights of the graph as a gaussian function of the distance between the corresponding rows of X, which represents an embedding of self

Parameters  

X array of shape (self.V, dim):

the coordinate matrix of the embedding

sigma=0, float: the parameter of the gaussian function:

Notes

When sigma == 0, the following value is used:

\[ \text{sigma} = \sqrt{\text{mean}(||X[self.edges[:, 0], :]-X[self.edges[:, 1], :]||^2)} \]
**set_weights** *(weights)*)
Set edge weights

**Parameters**  
weights: array :
array shape(self.V): edges weights

**show** *(X=None, ax=None)*)
Plots the current graph in 2D

**Parameters**  
X: None or array of shape (self.V, 2)  
a set of coordinates that can be used to embed the vertices in 2D. If X.shape[1]>2, a svd reduces X for display. By default, the graph is presented on a circle

ax: None or int, optional :
ax handle

**Returns**  
ax: axis handle :

**Notes**
This should be used only for small graphs.

**subfield** *(valid)*)
Returns a subfield of self, with only vertices such that valid > 0

**Parameters**  
valid: array of shape (self.V), :
nonzero for vertices to be retained

**Returns**  
F: Field instance, :
the desired subfield of self

**Notes**
The vertices are renumbered as [1..p] where p = sum(valid>0) when sum(valid) == 0 then None is returned

**subgraph** *(valid)*)
Creates a subgraph with the vertices for which valid>0 and with the corresponding set of edges

**Parameters**  
valid, array of shape (self.V): nonzero for vertices to be retained :

**Returns**  
G, WeightedGraph instance, the desired subgraph of self :

**Notes**
The vertices are renumbered as [1..p] where p = sum(valid>0) when sum(valid==0) then None is returned

**symmeterize** ()
Symmeterize self, modify edges and weights so that self.adjacency becomes the symmetric part of the current self.adjacency.

**threshold_bifurcations** *(refdim=0, th=-inf]*)
Analysis of the level sets of the field: Bifurcations are defined as changes in the topology in the level sets when the level (threshold) is varied This can be thought of as a kind of Morse analysis

**Parameters**  
 th: float, optional,
threshold so that only values above th are considered

**Returns**

- **idx**: array of shape (nlsets):
  - indices of the vertices that are local maxima

- **height**: array of shape (nlsets):
  - the depth of the local maxima depth[idx[i]] = q means that idx[i] is a q-order maximum
  - Note that this is also the diameter of the basins associated with local maxima

- **parents**: array of shape (nlsets):
  - the label of the maximum which dominates each local maximum i.e. it describes the hierarchy of the local maxima

- **label**: array of shape (self.V):
  - a labelling of the vertices according to their bassin

**to_coo_matrix**()

Return adjacency matrix as coo sparse

**Returns**

- **sp**: scipy.sparse matrix instance:
  - that encodes the adjacency matrix of self

**voronoi_diagram**(seeds, samples)

Defines the graph as the Voronoi diagram (VD) that links the seeds. The VD is defined using the sample points.

**Parameters**

- **seeds**: array of shape (self.V, dim):
- **samples**: array of shape (nsamples, dim):

**Notes**

By default, the weights are a Gaussian function of the distance The implementation is not optimal

**voronoi_labelling**(seed)

Performs a voronoi labelling of the graph

**Parameters**

- **seed**: array of shape (nseeds), type (np.int), :
  - vertices from which the cells are built

**Returns**

- **labels**: array of shape (self.V) the labelling of the vertices :

**ward**(nbcluster)

Ward’s clustering of self

**Parameters**

- **nbcluster**: int, :
  - the number of desired clusters

**Returns**

- **label**: array of shape (self.V):
  - the resulting field label

- **J** (float): the resulting inertia :
32.4 Functions

`nipy.algorithms.graph.field.field_from_coo_matrix_and_data(x, data)`
Instantiates a weighted graph from a (sparse) coo_matrix

Parameters

- `x`: (V, V) scipy.sparse.coo_matrix instance,
  the input matrix
- `data`: array of shape (V, dim),
  the field data

Returns

`ifield`: resulting Field instance:

`nipy.algorithms.graph.field.field_from_graph_and_data(g, data)`
Instantiates a Field from a WeightedGraph plus some feature data

Parameters

- `g`: (V, V) scipy.sparse.coo_matrix instance,
  the input matrix
- `data`: array of shape (V, dim),
  the field data

Returns

`ifield`: resulting field instance:
CHAPTER
THIRTYTHREE

ALGORITHMS.GRAPH.FOREST

33.1 Module: algorithms.graph.forest

Inheritance diagram for nipy.algorithms.graph.forest:

```
graph.graph.Graph ➔ graph.graph.WeightedGraph ➔ graph.forest.Forest
```

Module implements the Forest class

A Forest is a graph with a hierarchical structure. Each connected component of a forest is a tree. The main characteristic is that each node has a single parent, so that a Forest is fully characterized by a “parent” array, that defines the unique parent of each node. The directed relationships are encoded by the weight sign.

Note that some methods of WeightedGraph class (e.g. dijkstra’s algorithm) require positive weights, so that they cannot work on forests in the current implementation. Specific methods (e.g. all_sidtance()) have been set instead.

Main author: Bertrand thirion, 2007-2011

33.2 Forest

```python
class nipy.algorithms.graph.forest.Forest(V, parents=None):
    Bases: nipy.algorithms.graph.graph.WeightedGraph

    Forest structure, i.e. a set of trees

    The nodes can be segmented into trees.

    Within each tree a node has one parent and children that describe the associated hierarchical structure. Some of the nodes can be viewed as leaves, other as roots

    The edges within a tree are associated with a weight:
    •+1 from child to parent
    •-1 from parent to child
```
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>int</td>
<td>int &gt; 0, the number of vertices</td>
</tr>
<tr>
<td>E</td>
<td>int</td>
<td>the number of edges</td>
</tr>
<tr>
<td>parents</td>
<td>(self.V,)</td>
<td>array</td>
</tr>
<tr>
<td>edges</td>
<td>(self.E, 2)</td>
<td>array representing pairwise neighbors</td>
</tr>
<tr>
<td>weights</td>
<td>(self.E,)</td>
<td>+1/-1 for ascending/descending links</td>
</tr>
<tr>
<td>children</td>
<td>list</td>
<td>list of arrays that represents the children any node</td>
</tr>
</tbody>
</table>

Methods

- adjacency(): returns the adjacency matrix of the graph as a sparse coo matrix
- all_distances([seed]): returns all the distances of the graph as a tree
- anti_symmeterize(): anti-symmetricize self, i.e. produces the graph
- cc(): Compute the different connected components of the graph.
- check(): Check that self is indeed a forest, i.e.
- cliques(): Extraction of the graph's cliques
- compact_neigh(): returns a compact representation of self
- compute_children(): Define the children of each node (stored in self.children)
- copy(): returns a copy of self
- cut_redundancies(): Returns a graph with redundant edges removed; each edge (ab) is present only once in the graph.
- degrees(): defines the edge and weights array
- degrees(): Returns the degree of the graph vertices.
- depth_from_leaves(): compute an index for each node: 0 for the leaves, 1 for the others.
- dijkstra([seed]): Returns all the graph geodesic distances starting from seed
- floyd([seed]): Compute all the geodesic distances starting from seeds
- from_3d_grid(xyz[, k]): Sets the graph to be the topological neighbours graph
- get_E(): To get the number of edges in the graph
- get_V(): To get the number of vertices in the graph
- get_children([v]): Get the children of a node/each node
- get_descendants([v[, exclude_self]]): returns the nodes that are children of v as a list
- get_edges(): To get the graph’s edges
- get_vertices(): To get the graph’s vertices (as id)
- get_weights():
- is_connected(): States whether self is connected or not
- isleaf(): Identification of the leaves of the forest
- isroot(): Returns an indicator of nodes being roots
- kruskal(): Creates the Minimum Spanning Tree of self using Kruskal’s algo.
- leaves_of_a_subtree(ids[, custom]): tests whether the given nodes are the leaves of a certain subtree
- left_incidence(): Return left incidence matrix
- list_of_neighbors(): returns the set of neighbors of self as a list of arrays
- main_cc(): Returns the indexes of the vertices within the main cc
- merge_simple_branches(): Return a subforest, where chained branches are collapsed
- normalize([c]): Normalize the graph according to the index c
- propagate_upward(label): Propagation of a certain labelling from leaves to roots
- propagate_upward_and(prop): propagates from leaves to roots some binary property of the nodes
- remove_edges(valid): Removes all the edges for which valid==0
- remove_trivial_edges(): Removes trivial edges, i.e.
- reorder_from_leaves_to_roots(): reorder the tree so that the leaves come first then their
Table 33.1 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>right_incidence()</code></td>
<td>Return right incidence matrix</td>
</tr>
<tr>
<td><code>set_edges(edges)</code></td>
<td>Sets the graph’s edges</td>
</tr>
<tr>
<td><code>set_euclidian(X)</code></td>
<td>Compute the weights of the graph as the distances between the</td>
</tr>
<tr>
<td><code>set_gaussian(X[, sigma])</code></td>
<td>Compute the weights of the graph as a gaussian function</td>
</tr>
<tr>
<td><code>set_weights(weights)</code></td>
<td>Set edge weights</td>
</tr>
<tr>
<td><code>show([X, ax])</code></td>
<td>Plots the current graph in 2D</td>
</tr>
<tr>
<td><code>subforest(valid)</code></td>
<td>Creates a subforest with the vertices for which valid &gt; 0</td>
</tr>
<tr>
<td><code>subgraph(valid)</code></td>
<td>Creates a subgraph with the vertices for which valid&gt;0</td>
</tr>
<tr>
<td><code>symmetrize()</code></td>
<td>Symmetrize self, modify edges and weights so that</td>
</tr>
<tr>
<td><code>to_coo_matrix()</code></td>
<td>Return adjacency matrix as coo sparse</td>
</tr>
<tr>
<td><code>tree_depth()</code></td>
<td>Returns the number of hierarchical levels in the tree</td>
</tr>
<tr>
<td><code>voronoi_diagram(seeds, samples)</code></td>
<td>Defines the graph as the Voronoi diagram (VD)</td>
</tr>
<tr>
<td><code>voronoi_labelling(seed)</code></td>
<td>Performs a voronoi labelling of the graph</td>
</tr>
</tbody>
</table>

__init__ (V, parents=None)
Constructor

**Parameters**
- **V**: int
  the number of edges of the graph
- **parents**: None or (V,) array
  the parents of each vertex. If `parents=None`, the parents are set to range(V), i.e. each node is its own parent, and each node is a tree

**adjacency()**
returns the adjacency matrix of the graph as a sparse coo matrix

**Returns**
- **adj**: scipy.sparse matrix instance, :
  that encodes the adjacency matrix of self

**all_distances (seed=None)**
returns all the distances of the graph as a tree

**Parameters**
- **seed=None array of shape(nbseed) with values in [0..self.V-1]**:
  set of vertices from which the distances are computed

**Returns**
- **dg**: array of shape(nseed, self.V), the resulting distances :

**Notes**

By convention infinite distances are given the distance np.inf

**anti_symmetrize()**
anti-symmetrize self, i.e. produces the graph whose adjacency matrix would be the antisymmetric part of its current adjacency matrix

**cc()**
Compute the different connected components of the graph.

**Returns**
- **label**: array of shape(self.V), labelling of the vertices :

**check()**
Check that self is indeed a forest, i.e. contains no loop

**Returns**
a boolean b=0 iff there are loops, 1 otherwise :
Notes

Slow implementation, might be rewritten in C or cython

cliques()
Extraction of the graph cliques these are defined using replicator dynamics equations

    Returns  cliques: array of shape (self.V), type (np.int) :
               labelling of the vertices according to the clique they belong to

compact_neighb()
returns a compact representation of self

    Returns  idx: array of of shape(self.V + 1) :
               the positions where to find the neighbors of each node within neighb and weights

    neighb: array of shape(self.E), concatenated list of neighbors :

    weights: array of shape(self.E), concatenated list of weights :

compute_children()
Define the children of each node (stored in self.children)

copy()
returns a copy of self

cut_redundancies()
Returns a graph with redundant edges removed: each edge (ab) is present only once in the edge matrix: the corresponding weights are added.

    Returns  the resulting WeightedGraph :

define_graph_attributes()
define the edge and weights array

degrees()
Returns the degree of the graph vertices.

    Returns  rdegree: (array, type=int, shape=(self.V,)), the right degrees :
               ldegree: (array, type=int, shape=(self.V,)), the left degrees :

depth_from_leaves()
compute an index for each node: 0 for the leaves, 1 for their parents etc. and maximal for the roots.

    Returns  depth: array of shape (self.V): the depth values of the vertices :

dijkstra(seed=0)
Returns all the [graph] geodesic distances starting from seed x Parameters ———— seed (int, >-1, <self.V) or array of shape(p)

    edge(s) from which the distances are computed

    Returns  dg: array of shape (self.V), :
               the graph distance dg from ant vertex to the nearest seed

Notes

It is mandatory that the graph weights are non-negative
**floyd**(seed=None)

Compute all the geodesic distances starting from seeds

**Parameters**

- **seed=None**: array of shape (nbseed), type np.int
  
  vertex indexes from which the distances are computed if seed==None, then every edge
  is a seed point

**Returns**

- **dg array of shape (nbseed, self.V)**:
  
  the graph distance dg from each seed to any vertex

**Notes**

It is mandatory that the graph weights are non-negative. The algorithm proceeds by repeating Dijkstra’s algo for each seed. Floyd’s algo is not used (O(self.V)^3 complexity...)

**from_3d_grid**(xyz, k=18)

Sets the graph to be the topological neighbours graph of the three-dimensional coordinates set xyz, in the k-connectivity scheme

**Parameters**

- **xyz**: array of shape (self.V, 3) and type np.int
  
  - **k = 18**: the number of neighbours considered. (6, 18 or 26):

**Returns**

- **E(int)**: the number of edges of self:

**get_E()**

To get the number of edges in the graph

**get_V()**

To get the number of vertices in the graph

**get_children**(v=-1)

Get the children of a node/each node

**Parameters**

- **v**: int, optional
  
  a node index

**Returns**

- **children**: list of int the list of children of node v (if v is provided):
  
  a list of lists of int, the children of all nodes otherwise

**get_descendants**(v, exclude_self=False)

returns the nodes that are children of v as a list

**Parameters**

- **v**: int, a node index

**Returns**

- **desc**: list of int, the list of all descendant of the input node:

**get_edges()**

To get the graph’s edges

**get_vertices()**

To get the graph’s vertices (as id)

**get_weights()**

**is_connected()**

States whether self is connected or not

**isleaf()**

Identification of the leaves of the forest
Returns leaves: bool array of shape(self.V), indicator of the forest’s leaves:

isroot()  
Returns an indicator of nodes being roots

Returns roots: array of shape(self.V), indicator of the forest’s roots:

kruskal()  
Creates the Minimum Spanning Tree of self using Kruskal’s algo. Efficient is self is sparse

Returns K, WeightedGraph instance: the resulting MST:

Notes

If self contains several connected components, will have the same number k of connected components

leaves_of_a_subtree(ids, custom=False)  
tests whether the given nodes are the leaves of a certain subtree

Parameters ids: array of shape (n) that takes values in [0..self.V-1]:

custom == False, boolean:
   if custom==true the behavior of the function is more specific - the different connected components are considered as being in a same greater tree - when a node has more than two subbranches, any subset of these children is considered as a subtree

left_incidence()  
Return left incidence matrix

Returns left_incid: list:
   the left incidence matrix of self as a list of lists: i.e. the list[[e.0.0, .., e.0.i(0)], .., [e.V.0, E.V.i(V)]] where e.i.j is the set of edge indexes so that e.i.j[0] = i

list_of_neighbors()  
returns the set of neighbors of self as a list of arrays

main_cc()  
Returns the indexes of the vertices within the main cc

Returns idx: array of shape (sizeof main cc):

merge_simple_branches()  
Return a subforest, where chained branches are collapsed

Returns sf, Forest instance, same as self, without any chain:

normalize(c=0)  
Normalize the graph according to the index c Normalization means that the sum of the edges values that go into or out each vertex must sum to 1

Parameters c=0 in {0, 1, 2}, optional: index that designates the way:
   according to which D is normalized c == 0 => for each vertex a, sum{edge[e, 0]=a} D[e]=1 c == 1 => for each vertex b, sum{edge[e, 1]=b} D[e]=1 c == 2 => symmetric (‘l2’) normalization

Notes

Note that when sum_{edge[e, .] == a } D[e] = 0, nothing is performed
propagate_upward (label)
Propagates a certain labelling from leaves to roots. Assuming that label is a certain positive integer field this propagates these labels to the parents whenever the children nodes have coherent properties otherwise the parent value is unchanged.

Parameters
label: array of shape(self.V)

Returns
label: array of shape(self.V)

propagate_upward_and (prop)
Propagates from leaves to roots some binary property of the nodes so that prop[parents] = logical_and(prop[children])

Parameters
prop, array of shape(self.V), the input property:

Returns
prop, array of shape(self.V), the output property field:

remove_edges (valid)
Removes all the edges for which valid==0.

Parameters
valid: (self.E,) array

remove_trivial_edges()
Removes trivial edges, i.e. edges that are (vv)-like. self.weights and self.E are corrected accordingly.

Returns
self.E (int): The number of edges:

reorder_from_leaves_to_roots()
reorder the tree so that the leaves come first then their parents and so on, and the roots are last.

Returns
order: array of shape(self.V):
the order of the old vertices in the reordered graph

right_incidence()
Return right incidence matrix

Returns
right_incid: list:
the right incidence matrix of self as a list of lists: i.e. the list[[e.0.0, ... e.0.i(0)], ... [e.V.0, E.V.i(V)]] where e.i.j is the set of edge indexes so that e.i.j[1] = i

set_edges (edges)
Sets the graph’s edges

set_euclidian (X)
Compute the weights of the graph as the distances between the corresponding rows of X, which represents an embedding of self

Parameters
X array of shape (self.V, edim), :
the coordinate matrix of the embedding

set_gaussian (X, sigma=0)
Compute the weights of the graph as a gaussian function of the distance between the corresponding rows of X, which represents an embedding of self.

Parameters
X array of shape (self.V, dim):
the coordinate matrix of the embedding

sigma=0, float: the parameter of the gaussian function:
Notes

When sigma == 0, the following value is used:

\[
\text{sigma} = \sqrt{\text{mean}(||X[\text{self.edges}[0,0]:]-X[\text{self.edges}[0,1]:]|^2)}
\]

set_weights(weights)

Set edge weights

Parameters weights: array

array shape(self.V): edges weights

show(X=None, ax=None)

Plots the current graph in 2D

Parameters X: None or array of shape (self.V, 2)

a set of coordinates that can be used to embed the vertices in 2D. If X.shape[1]>2, a svd
reduces X for display. By default, the graph is presented on a circle

ax: None or int, optional

ax handle

Returns ax: axis handle:

Notes

This should be used only for small graphs.

subforest(valid)

Creates a subforest with the vertices for which valid > 0

Parameters valid: array of shape (self.V): indicator of the selected nodes

Returns subforest: a new forest instance, with a reduced set of nodes:

Notes

The children of deleted vertices become their own parent

subgraph(valid)

Creates a subgraph with the vertices for which valid>0 and with the correponding set of edges

Parameters valid, array of shape (self.V): nonzero for vertices to be retained

Returns G, WeightedGraph instance, the desired subgraph of self:

Notes

The vertices are renumbered as [1..p] where p = sum(valid>0) when sum(valid==0) then None is returned

symmeterize()

Symmeterize self, modify edges and weights so that self.adjacency becomes the symmetric part of the current self.adjacency.

to_coo_matrix()

Return adjacency matrix as coo sparse

Returns sp: scipy.sparse matrix instance:
that encodes the adjacency matrix of self

```
  **tree_depth()**
  Returns the number of hierarchical levels in the tree

  **voronoi_diagram(seeds, samples)**
  Defines the graph as the Voronoi diagram (VD) that links the seeds. The VD is defined using the sample points.

    Parameters
    seeds: array of shape (self.V, dim):
    samples: array of shape (nsamples, dim):

  **Notes**

  By default, the weights are a Gaussian function of the distance. The implementation is not optimal

  **voronoi_labelling(seed)**
  Performs a voronoi labelling of the graph

    Parameters
    seed: array of shape (nseeds), type (np.int), :
    vertices from which the cells are built

    Returns
    labels: array of shape (self.V) the labelling of the vertices:
34.1 Module: `algorithms.graph.graph`

Inheritance diagram for `nipy.algorithms.graph.graph`:

```
graph.graph.Graph    graph.graph.WeightedGraph
```

This module implements two graph classes:
- **Graph**: basic topological graph, i.e. vertices and edges. This kind of object only has topological properties.
- **WeightedGraph** (Graph): also has a value associated with edges, called weights, that are used in some computational procedures (e.g. path length computation). Importantly these objects are equivalent to square sparse matrices, which is used to perform certain computations.

This module also provides several functions to instantiate WeightedGraphs from data:
- k nearest neighbours (where samples are rows of a 2D-array)
- epsilon-neighbors (where sample rows of a 2D-array)
- representation of the neighbors on a 3d grid (6-, 18- and 26-neighbors)
- Minimum Spanning Tree (where samples are rows of a 2D-array)

Author: Bertrand Thirion, 2006–2011

34.2 Classes

34.2.1 Graph

```python
class nipy.algorithms.graph.graph.Graph(V, E=0, edges=None)
Bases: object
```

Basic topological (non-weighted) directed Graph class

Member variables:
- V (int > 0): the number of vertices
- E (int >= 0): the number of edges
Properties:

- vertices (list, type=int, shape=(V,)) vertices id
- edges (list, type=int, shape=(E,2)): edges as vertices id tuples

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>adjacency()</td>
<td>returns the adjacency matrix of the graph as a sparse coo matrix</td>
</tr>
<tr>
<td>cc()</td>
<td>Compute the different connected components of the graph.</td>
</tr>
<tr>
<td>degrees()</td>
<td>Returns the degree of the graph vertices.</td>
</tr>
<tr>
<td>get_E()</td>
<td>To get the number of edges in the graph</td>
</tr>
<tr>
<td>get_V()</td>
<td>To get the number of vertices in the graph</td>
</tr>
<tr>
<td>get_edges()</td>
<td>To get the graph’s edges</td>
</tr>
<tr>
<td>get_vertices()</td>
<td>To get the graph’s vertices (as id)</td>
</tr>
<tr>
<td>main_cc()</td>
<td>Returns the indexes of the vertices within the main cc</td>
</tr>
<tr>
<td>set_edges(edges)</td>
<td>Sets the graph’s edges</td>
</tr>
<tr>
<td>show(ax)</td>
<td>Shows the graph as a planar one.</td>
</tr>
<tr>
<td>to_coo_matrix()</td>
<td>Return adjacency matrix as coo sparse</td>
</tr>
</tbody>
</table>

__init__ (V, E=0, edges=None)

Constructor

Parameters

- V: int
  - the number of vertices
- E: int, optional
  - the number of edges
- edges: None or shape (E, 2) array, optional
  - edges of graph

adjacency()

returns the adjacency matrix of the graph as a sparse coo matrix

Returns

adj: scipy.sparse matrix instance,
- that encodes the adjacency matrix of self

cc()

Compute the different connected components of the graph.

Returns

label: array of shape(self.V), labelling of the vertices:

degrees()

Returns the degree of the graph vertices.

Returns

rdegree: (array, type=int, shape=(self.V,)), the right degrees:
- ldegree: (array, type=int, shape=(self.V,)), the left degrees:

get_E()

To get the number of edges in the graph

get_V()

To get the number of vertices in the graph
get_edges()  
To get the graph’s edges

get_vertices()  
To get the graph’s vertices (as id)

main_cc()  
Returns the indexes of the vertices within the main cc

    Returns  idx: array of shape (sizeof main cc):

set_edges(edges)  
Sets the graph’s edges

show(ax=None)  
Shows the graph as a planar one.

    Parameters  ax, axis handle:

    Returns  ax, axis handle:

to_coo_matrix()  
Return adjacency matrix as coo sparse

    Returns  sp: scipy.sparse matrix instance,:
                  that encodes the adjacency matrix of self

34.2.2 WeightedGraph

class nipy.algorithms.graph.graph.WeightedGraph(V, edges=None, weights=None)  
Bases: nipy.algorithms.graph.graph.Graph

Basic weighted, directed graph class

Member variables:

• V (int): the number of vertices
• E (int): the number of edges

Methods

• vertices (list, type=int, shape=(V,)): vertices id
• edges (list, type=int, shape=(E,2)): edges as vertices id tuples
• weights (list, type=int, shape=(E,)): weights / lengths of the graph’s edges

Methods

adjacency()  
returns the adjacency matrix of the graph as a sparse coo matrix

anti_symmeterize()  
anti-symmeterize self, i.e. produces the graph

cc()  
Compte the different connected components of the graph.

cliques()  
Extraction of the graph’s cliques

compact_neighb()  
returns a compact representation of self

copy()  
returns a copy of self

cut_redundancies()  
Returns a graph with redundant edges removed: each edge (ab) is present only once in the edge matrix.

degrees()  
Returns the degree of the graph vertices.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dijkstra([seed])</td>
<td>Returns all the graph geodesic distances starting from seed</td>
</tr>
<tr>
<td>floyd([seed])</td>
<td>Compute all the geodesic distances starting from seeds</td>
</tr>
<tr>
<td>from_3d_grid(xyz[, k])</td>
<td>Sets the graph to be the topological neighbours graph</td>
</tr>
<tr>
<td>get_E()</td>
<td>To get the number of edges in the graph</td>
</tr>
<tr>
<td>get_V()</td>
<td>To get the number of vertices in the graph</td>
</tr>
<tr>
<td>get_edges()</td>
<td>To get the graph’s edges</td>
</tr>
<tr>
<td>get_vertices()</td>
<td>To get the graph’s vertices (as id)</td>
</tr>
<tr>
<td>get_weights()</td>
<td></td>
</tr>
<tr>
<td>is_connected()</td>
<td>States whether self is connected or not</td>
</tr>
<tr>
<td>kruskal()</td>
<td>Creates the Minimum Spanning Tree of self using Kruskal’s algo.</td>
</tr>
<tr>
<td>left_incidence()</td>
<td>Return left incidence matrix</td>
</tr>
<tr>
<td>list_of_neighbors()</td>
<td>returns the set of neighbors of self as a list of arrays</td>
</tr>
<tr>
<td>main_cc()</td>
<td>Returns the indexes of the vertices within the main cc</td>
</tr>
<tr>
<td>normalize([c])</td>
<td>Normalize the graph according to the index c</td>
</tr>
<tr>
<td>remove_edges(valid)</td>
<td>Removes all the edges for which valid==0</td>
</tr>
<tr>
<td>remove_trivial_edges()</td>
<td>Removes trivial edges, i.e.</td>
</tr>
<tr>
<td>right_incidence()</td>
<td>Return right incidence matrix</td>
</tr>
<tr>
<td>set_edges(edges)</td>
<td>Sets the graph’s edges</td>
</tr>
<tr>
<td>set_euclidian(X)</td>
<td>Compute the weights of the graph as the distances between the</td>
</tr>
<tr>
<td>set_gaussian(X[, sigma])</td>
<td>Compute the weights of the graph as a gaussian function</td>
</tr>
<tr>
<td>set_weights(weights)</td>
<td>Set edge weights</td>
</tr>
<tr>
<td>show([X, ax])</td>
<td>Plots the current graph in 2D</td>
</tr>
<tr>
<td>subgraph(valid)</td>
<td>Creates a subgraph with the vertices for which valid&gt;0</td>
</tr>
<tr>
<td>symmeterize()</td>
<td>Symmeterize self, modify edges and weights so that</td>
</tr>
<tr>
<td>to_coo_matrix()</td>
<td>Return adjacency matrix as coo sparse</td>
</tr>
<tr>
<td>voronoi_diagram(seeds, samples)</td>
<td>Defines the graph as the Voronoi diagram (VD)</td>
</tr>
<tr>
<td>voronoi_labelling(seed)</td>
<td>Performs a voronoi labelling of the graph</td>
</tr>
</tbody>
</table>

```python
__init__(V, edges=None, weights=None)
```

Constructor

**Parameters**  

- **V** : int  
  (int > 0) the number of vertices
- **edges** : (E, 2) array, type int  
  edges of the graph
- **weights** : (E, 2) array, type=int  
  weights/lenghts of the edges

**adjacency()**

returns the adjacency matrix of the graph as a sparse coo matrix

**Returns**  

- **adj** : scipy.sparse matrix instance,  
  that encodes the adjacency matrix of self

**anti_symmeterize()**

anti-symmeterize self, i.e. produces the graph whose adjacency matrix would be the antisymmetric part of its current adjacency matrix

**cc()**

Compute the different connected components of the graph.
Returns label: array of shape(self.V), labelling of the vertices:

cliques()
Extraction of the graph cliques these are defined using replicator dynamics equations

Returns cliques: array of shape (self.V), type (np.int):
labelling of the vertices according to the clique they belong to

compact_neighb()
returns a compact representation of self

Returns idx: array of shape(self.V + 1):
the positions where to find the neighbors of each node within neighb and weights

neighb: array of shape(self.E), concatenated list of neighbors:
weights: array of shape(self.E), concatenated list of weights:

copy()
returns a copy of self

cut_redundancies()
Returns a graph with redundant edges removed: each edge (ab) is present only once in the edge matrix: the corresponding weights are added.

Returns the resulting WeightedGraph:

degrees()
Returns the degree of the graph vertices.

Returns rdegree: (array, type=int, shape=(self.V,)), the right degrees:
ldegree: (array, type=int, shape=(self.V,)), the left degrees:

dijkstra (seed=0)
Returns all the graph geodesic distances starting from seed x Parameters ——— seed (int, >-1, <self.V)
or array of shape(p)
edge(s) from which the distances are computed

Returns dg: array of shape (self.V),:
the graph distance dg from any vertex to the nearest seed

Notes
It is mandatory that the graph weights are non-negative

floyd (seed=None)
Compute all the geodesic distances starting from seeds

Parameters seed=None: array of shape (nbseed), type np.int:
vertex indexes from which the distances are computed if seed==None, then every edge
is a seed point

Returns dg array of shape (nbseed, self.V):
the graph distance dg from each seed to any vertex
Notes

It is mandatory that the graph weights are non-negative. The algorithm proceeds by repeating Dijkstra’s algo for each seed. Floyd’s algo is not used (O(self.V)^3 complexity...)

```
from 3d_grid(xyz, k=18)
```

Sets the graph to be the topological neighbours graph of the three-dimensional coordinates set xyz, in the k-connectivity scheme.

- **Parameters**
  - `xyz`: array of shape (self.V, 3) and type np.int,
  - `k = 18`: the number of neighbours considered. (6, 18 or 26)

- **Returns**
  - `E(int)`: the number of edges of self

```
get_E()
```

To get the number of edges in the graph.

```
get_V()
```

To get the number of vertices in the graph.

```
get_edges()
```

To get the graph’s edges.

```
get_vertices()
```

To get the graph’s vertices (as id).

```
get_weights()
```

```
is_connected()
```

States whether self is connected or not.

```
kruskal()
```

Creates the Minimum Spanning Tree of self using Kruskal’s algo. efficient is self is sparse.

- **Returns**
  - `K, WeightedGraph instance`: the resulting MST

Notes

If self contains several connected components, will have the same number k of connected components.

```
left_incidence()
```

Return left incidence matrix.

- **Returns**
  - `left_incid: list`

    the left incidence matrix of self as a list of lists: i.e. the list[[e.0.0, .., e.0.i(0)], .., [e.V.0, E.V.i(V)]] where e.i.j is the set of edge indexes so that e.i.j[0] = i

```
list_of_neighbors()
```

returns the set of neighbors of self as a list of arrays.

```
main_cc()
```

Returns the indexes of the vertices within the main cc.

- **Returns**
  - `idx: array of shape (sizeof main cc)`

```
normalize(c=0)
```

Normalize the graph according to the index c Normalization means that the sum of the edges values that go into or out each vertex must sum to 1.

- **Parameters**
  - `c=0 in {0, 1, 2}`, optional: index that designates the way.
according to which $D$ is normalized

$c == 0$ => for each vertex $a$, $\sum_{\text{edge}[e, 0]=a} D[e] = 1$
$c == 1$ => for each vertex $b$, $\sum_{\text{edge}[e, 1]=b} D[e] = 1$  
$c == 2$ => symmetric ('l2') normalization

**Notes**

Note that when $\sum_{\text{edge}[e, .,] = a} D[e] = 0$, nothing is performed

**remove_edges** *(valid)*

Removes all the edges for which valid==0

**Parameters**
- **valid**: (self.E,) array

**remove_trivial_edges** *

Removes trivial edges, i.e. edges that are (vv)-like self.weights and self.E are corrected accordingly

**Returns**
- **self.E** (int): The number of edges

**right_incidence** *

Return right incidence matrix

**Returns**
- **right_inc**: list
  - the right incidence matrix of self as a list of lists: i.e. the list[[e.0.0, ... e.0.i(0)], ..., [e.V.0, E.V.i(V)]] where e.i is the set of edge indexes so that e.i[1] = i

**set_edges** *(edges)*

Sets the graph’s edges

**set_euclidian** *(X)*

Compute the weights of the graph as the distances between the corresponding rows of $X$, which represents an embedding of self

**Parameters**
- **X** array of shape (self.V, edim),:
  - the coordinate matrix of the embedding

**set_gaussian** *(X, sigma=0)*

Compute the weights of the graph as a gaussian function of the distance between the corresponding rows of $X$, which represents an embedding of self

**Parameters**
- **X** array of shape (self.V, dim):
  - the coordinate matrix of the embedding
- **sigma**=0, float: the parameter of the gaussian function

**Notes**

When $sigma == 0$, the following value is used: $sigma = \sqrt{\text{mean}||X[\text{self.edges[:, 0]}, :] - X[\text{self.edges[:, 1]}, :||^2))}$

**set_weights** *(weights)*

Set edge weights

**Parameters**
- **weights**: array
  - array shape(self.V): edges weights

**show** *(X=None, ax=None)*

Plots the current graph in 2D
Parameters  

- **X**: None or array of shape (self.V, 2)
  - a set of coordinates that can be used to embed the vertices in 2D. If X.shape[1]>2, a svd reduces X for display. By default, the graph is presented on a circle.

- **ax**: *None or int, optional*
  - ax handle

Returns  

- **ax**: axis handle

Notes

This should be used only for small graphs.

**subgraph** *(valid)*

Creates a subgraph with the vertices for which valid>0 and with the corresponding set of edges.

Parameters  

- **valid**, array of shape (self.V): nonzero for vertices to be retained

Returns  

- **G**, WeightedGraph instance, the desired subgraph of self

Notes

The vertices are renumbered as [1..p] where p = sum(valid>0) when sum(valid==0) then None is returned.

**symmeterize**

Symmeterize self, modify edges and weights so that self.adjacency becomes the symmetric part of the current self.adjacency.

**to_coo_matrix**

Return adjacency matrix as coo sparse.

Returns  

- **sp**: scipy.sparse matrix instance
  - that encodes the adjacency matrix of self

**voronoi_diagram** *(seeds, samples)*

Defines the graph as the Voronoi diagram (VD) that links the seeds. The VD is defined using the sample points.

Parameters  

- **seeds**: array of shape (self.V, dim)
- **samples**: array of shape (nsamples, dim)

Notes

By default, the weights are a Gaussian function of the distance. The implementation is not optimal.

**voronoi_labelling** *(seed)*

Performs a voronoi labelling of the graph.

Parameters  

- **seed**: array of shape (nseeds), type (np.int)
  - vertices from which the cells are built

Returns  

- **labels**: array of shape (self.V) the labelling of the vertices
34.3 Functions

*nipy.algorithms.graph.graph.complete_graph*(n)
returns a complete graph with n vertices

*nipy.algorithms.graph.graph.concatenate_graphs*(G1, G2)
Returns the concatenation of the graphs G1 and G2. It is thus assumed that the vertices of G1 and G2 represent disjoint sets.

Parameters  
G1, G2: the two WeightedGraph instances to be concatenated:

Returns  
G, WeightedGraph, the concatenated graph:

Notes

This implies that the vertices of G corresponding to G2 are labeled [G1.V .. G1.V+G2.V]

*nipy.algorithms.graph.graph.eps_nn*(X, eps=1.0)
Returns the eps-nearest-neighbours graph of the data.

Parameters  
X, array of shape (n_samples, n_features), input data:

eps, float, optional: the neighborhood width:

Returns  
the resulting graph instance:

*nipy.algorithms.graph.graph.graph_3d_grid*(xyz, k=18)
Utility that computes the six neighbors on a 3d grid.

Parameters  
xyz: array of shape (n_samples, 3); grid coordinates of the points:

k: neighboring system, equal to 6, 18, or 26:

Returns  
i, j, d 3 arrays of shape (E),:

where E is the number of edges in the resulting graph (i, j) represent the edges, d their weights

*nipy.algorithms.graph.graph.knn*(X, k=1)
returns the k-nearest-neighbours graph of the data.

Parameters  
X, array of shape (n_samples, n_features): the input data:

k, int, optional: is the number of neighbours considered:

Returns  
the corresponding WeightedGraph instance:

Notes

The knn system is symmeterized: if (ab) is one of the edges then (ba) is also included

*nipy.algorithms.graph.graph.lil_cc*(lil)
Returns the connected comonents of a graph represented as a list of lists.

Parameters  
lil: a list of list representing the graph neighbors:

Returns  
label a vector of shape len(lil): connected components labelling:
Notes

Dramatically slow for non-sparse graphs

\texttt{nipy.algorithms.graph.graph.mst(X)}

Returns the WeightedGraph that is the minimum Spanning Tree of X

**Parameters**

\texttt{X: data array, of shape(n_samples, n_features)}:

**Returns**

the corresponding WeightedGraph instance:

\texttt{nipy.algorithms.graph.graph.wgraph_from_3d_grid(xyz, k=18)}

Create graph as the set of topological neighbours of the three-dimensional coordinates set xyz, in the k-connectivity scheme

**Parameters**

\texttt{xyz: array of shape (nsamples, 3) and type np.int, :}

\texttt{k = 18: the number of neighbours considered. (6, 18 or 26)}:

**Returns**

the WeightedGraph instance:

\texttt{nipy.algorithms.graph.graph.wgraph_from_adjacency(x)}

Instantiates a weighted graph from a square 2D array

**Parameters**

\texttt{x: 2D array instance, the input array}:

**Returns**

\texttt{wg: WeightedGraph instance}:

\texttt{nipy.algorithms.graph.graph.wgraph_from_coo_matrix(x)}

Instantiates a weighted graph from a (sparse) coo_matrix

**Parameters**

\texttt{x: scipy.sparse.coo_matrix instance, the input matrix}:

**Returns**

\texttt{wg: WeightedGraph instance}:
35.1 Module: algorithms.interpolation

Inheritance diagram for nipy.algorithms.interpolation:

```
algorithms.interpolation.ImageInterpolator
```

Image interpolators using ndimage.

35.2 ImageInterpolator

```python
class nipy.algorithms.interpolation.ImageInterpolator(image, order=3)
    Bases: object
    
    Interpolate Image instance at arbitrary points in world space
    The resampling is done with scipy.ndimage.
```

Methods

```
evaluate(points)  # Resample image at points in world space

__init__(image, order=3)

Parameters
    image : Image
        Image to be interpolated
    order : int, optional
        order of spline interpolation as used in scipy.ndimage. Default is 3.
```
evaluate(points)
Resample image at points in world space

Parameters
points : array
values in self.image.coordmap.output_coords. Each row is a point.

Returns
V : ndarray
interpolator of self.image evaluated at points
36.1 Module: `algorithms.kernel_smooth`

Inheritance diagram for `nipy.algorithms.kernel_smooth`:

```
algorithms.kernel_smooth.LinearFilter
```

Linear filter(s). For the moment, only a Gaussian smoothing filter

36.2 Class

36.3 LinearFilter

```python
class nipy.algorithms.kernel_smooth.LinearFilter(coordmap, shape, fwhm=6.0, scale=1.0, location=0.0, cov=None)
```

**Bases:** object

A class to implement some FFT smoothers for Image objects. By default, this does a Gaussian kernel smooth. More choices would be better!

**Methods**

```
_call__(X[, axis])          Compute kernel from points
smooth(inimage[, clean, is_fft])  Apply smoothing to inimage
```

```
__init__(coordmap, shape, fwhm=6.0, scale=1.0, location=0.0, cov=None)
```

**Parameters**

- `coordmap`: CoordinateMap
- `shape`: sequence
fwhm : float, optional
fwhm for Gaussian kernel, default is 6.0
scale : float, optional
scaling to apply to data after smooth, default 1.0
location : float
offset to apply to data after smooth and scaling, default 0
cov : None or array, optional
Covariance matrix

normalization = ‘l1sum’

smooth(inimage, clean=False, is_fft=False)
Apply smoothing to inimage

Parameters
inimage : Image
The image to be smoothed. Should be 3D.
clean : bool, optional
Should we call nan_to_num on the data before smoothing?
is_fft : bool, optional
Has the data already been fft’d?

Returns
s_image : Image
New image, with smoothing applied

36.4 Functions

nipy.algorithms.kernel_smooth.fwhm2sigma(fwhm)
Convert a FWHM value to sigma in a Gaussian kernel.

Parameters
fwhm : array-like
FWHM value or values

Returns
sigma : array or float
sigma values corresponding to fwhm values

Examples

>>> sigma = fwhm2sigma(6)

>>> sigmavae = fwhm2sigma([6, 7, 8])

>>> sigma == sigmavae[0]
True

nipy.algorithms.kernel_smooth.sigma2fwhm(sigma)
Convert a sigma in a Gaussian kernel to a FWHM value

Parameters
sigma : array-like
sigma value or values
Returns fwhm : array or float

fwhm values corresponding to sigma values

Examples

```python
>>> fwhm = sigma2fwhm(3)
>>> fwhms = sigma2fwhm([3, 4, 5])
>>> fwhm == fwhms[0]
True
```
37.1 Module: algorithms.optimize

nipy.algorithms.optimize.fmin_steepest(f, x0, fprime=None, xtol=0.0001, ftol=0.0001, maxiter=None, callback=None, disp=True)

Minimize a function using a steepest gradient descent algorithm. This complements the collection of minimization routines provided in scipy.optimize. Steepest gradient iterations are cheaper than in the conjugate gradient or Newton methods, hence convergence may sometimes turn out faster although more iterations are typically needed.

Parameters
- f : callable
  - Function to be minimized
- x0 : array
  - Starting point
- fprime : callable
  - Function that computes the gradient of f
- xtol : float
  - Relative tolerance on step sizes in line searches
- ftol : float
  - Relative tolerance on function variations
- maxiter : int
  - Maximum number of iterations
- callback : callable
  - Optional function called after each iteration is complete
- disp : bool
  - Print convergence message if True

Returns
- x : array
  - Gradient descent fix point, local minimizer of f
38.1 Module: algorithms.registration.affine

Inheritance diagram for `nipy.algorithms.registration.affine`:

38.2 Classes

38.2.1 Affine

```python
class nipy.algorithms.registration.affine.Affine(array=None, radius=100)
    Bases: nipy.algorithms.registration.transform.Transform

Methods

    apply(xyz)
    as_affine(dtype)
    compose(other) Compose this transform onto another
    copy()
    from_matrix44(aff) Convert a 4x4 matrix describing an affine transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for shearing when combined with non-unitary scales).
    inv() Return the inverse affine transform.

    __init__(array=None, radius=100)
    apply(xyz)
    as_affine(dtype='double')
```

273
compose\(\text{\texttt{(other)}}\)
Compose this transform onto another

**Parameters**
- \texttt{other} : Transform
  transform that we compose onto

**Returns**
- \texttt{composed_transform} : Transform
  a transform implementing the composition of self on \texttt{other}

copy()

from\texttt{matrix44}\(\text{\texttt{(aff)}}\)
Convert a 4x4 matrix describing an affine transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for shearing when combined with non-unitary scales).
In case the transform has a negative determinant, set the \_\texttt{direct} attribute to False.

inv()
Return the inverse affine transform.

**is\_direct**

param

param\_inds = \[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\]

pre\_rotation

precond

rotation

scaling

translation

### 38.2.2 Affine2D

class \texttt{nipy.algorithms.registration.affine.Affine2D}(array=None, radius=100)
Bases: \texttt{nipy.algorithms.registration.affine.Affine}

**Methods**

apply\(\text{\texttt{(xyz)}}\)

as\_affine\(\text{\texttt{[\texttt{dtype}]}}\)

compose\(\text{\texttt{(other)}}\) Compose this transform onto another

copy()

from\texttt{matrix44}\(\text{\texttt{(aff)}}\) Convert a 4x4 matrix describing an affine transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for shearing when combined with non-unitary scales).

inv() Return the inverse affine transform.

__init__\(\text{\texttt{(array=None, radius=100)}}\)

apply\(\text{\texttt{(xyz)}}\)

as\_affine\(\text{\texttt{[\texttt{dtype}]}}}\)

compose\(\text{\texttt{(other)}}\) Compose this transform onto another
Parameters  other : Transform

transform that we compose onto

Returns  composed_transform : Transform

a transform implementing the composition of self on other

\texttt{copy} ()

\texttt{from\_matrix44} (aff)

Convert a 4x4 matrix describing an affine transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for shearing when combined with non-unitary scales). In case the transform has a negative determinant, set the \_direct attribute to False.

\texttt{inv} ()

Return the inverse affine transform.

\texttt{is\_direct}

\texttt{param}

\texttt{param\_inds = [0, 1, 5, 6, 7, 11]}

\texttt{pre\_rotation}

\texttt{precond}

\texttt{rotation}

\texttt{scaling}

\texttt{translation}

38.2.3 Rigid

class nipy.algorithms.registration.affine.Rigid (array=None, radius=100)

Bases: nipy.algorithms.registration.affine.Affine

Methods

\texttt{apply} (xyz)

\texttt{as\_affine} (dtype)

\texttt{compose} (other)  

Compose this transform onto another

\texttt{copy} ()

\texttt{from\_matrix44} (aff)  

Convert a 4x4 matrix describing a rigid transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for shearing when combined with non-unitary scales). In case the transform has a negative determinant, set the \_direct attribute to False.

\texttt{inv} ()  

Return the inverse affine transform.

\texttt{__init__} (array=None, radius=100)

\texttt{apply} (xyz)

\texttt{as\_affine} (dtype='double')

\texttt{compose} (other)  

Compose this transform onto another

Parameters  other : Transform

transform that we compose onto
Returns `composed_transform`: Transform

a transform implementing the composition of self on other

copy()

`from_matrix44(aff)`

Convert a 4x4 matrix describing a rigid transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for pre-rotation when combined with non-unitary scales). In case the transform has a negative determinant, set the `_direct` attribute to False.

inv()

Return the inverse affine transform.

is_direct

pre_rotation

precond

rotation

scaling

translation

38.2.4 Rigid2D

class `nipy.algorithms.registration.affine.Rigid2D(array=None, radius=100)`

Bases: `nipy.algorithms.registration.affine.Rigid`

Methods

```python
apply(xyz)
as_affine(dtype)
compose(other)
copy()
from_matrix44(aff)
inv()
```

__init__(array=None, radius=100)

apply(xyz)

as_affine(dtype='double')

compose(other)

Compose this transform onto another transform that we compose onto

Parameters other : Transform

Returns composed_transform : Transform

a transform implementing the composition of self on other
copy()

from_matrix44(aff)
Convert a 4x4 matrix describing a rigid transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for pre-rotation when combined with non-unitary scales). In case the transform has a negative determinant, set the _direct attribute to False.

inv()
Return the inverse affine transform.

is_direct

param

param_inds = [0, 1, 5]

pre_rotation

precond

rotation

scaling

translation

38.2.5 Similarity

class nipy.algorithms.registration.affine.Similarity(array=None, radius=100)
    Bases: nipy.algorithms.registration.affine.Affine

Methods

apply(xyz)
as_affine(dtype)
compose(other) Compose this transform onto another
copy()
from_matrix44(aff) Convert a 4x4 matrix describing a similarity transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for pre-rotation when combined with non-unitary scales).
inv() Return the inverse affine transform.

__init__(array=None, radius=100)
apply(xyz)
as_affine(dtype=’double’)
compose(other)
    Compose this transform onto another
    Parameters other : Transform
    transform that we compose onto
    Returns composed_transform : Transform
    a transform implementing the composition of self on other
copy()
from_matrix44(aff)
Convert a 4x4 matrix describing a similarity transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for pre-rotation when combined with non-unitary scales). In case the transform has a negative determinant, set the _direct attribute to False.

inv()
Return the inverse affine transform.

is_direct
param
param_inds = [0, 1, 2, 3, 4, 5, 6]
pre_rotation
precond
rotation
scaling
translation

38.2.6 Similarity2D

class nipy.algorithms.registration.affine.Similarity2D(array=None, radius=100)
Bases: nipy.algorithms.registration.affine.Similarity

Methods

__init__ (array=None, radius=100)
apply(xyz)
as_affine([dtype])
compose(other) Compose this transform onto another
copy()
from_matrix44(aff) Convert a 4x4 matrix describing a similarity transform into a 12-sized vector of natural affine parameters: translation, rotation, log-scale, pre-rotation (to allow for pre-rotation when combined with non-unitary scales).
inv() Return the inverse affine transform.

Parameters other : Transform transform that we compose onto

Returns composed_transform : Transform a transform implementing the composition of self on other

copy()
from_matrix44(aff) Convert a 4x4 matrix describing a similarity transform into a 12-sized vector of natural affine parameters:
translation, rotation, log-scale, pre-rotation (to allow for pre-rotation when combined with non-unitary scales). In case the transform has a negative determinant, set the _direct attribute to False.

```python
inv()
    Return the inverse affine transform.

is_direct
param
param_inds = [0, 1, 5, 6]
pre_rotation
precond
rotation
scaling
translation
```

### 38.3 Functions

```python
nipy.algorithms.registration.affine.inverse_affine(affine)
nipy.algorithms.registration.affine.preconditioner(radius)
Computes a scaling vector pc such that, if p=(u,r,s,q) represents affine transformation parameters, where u is a translation, r and q are rotation vectors, and s is the vector of log-scales, then all components of (p/pc) are roughly comparable to the translation component.

To that end, we use a radius parameter which represents the ‘typical size’ of the object being registered. This is used to reformat the parameter vector (translation+rotation+scaling+pre-rotation) so that each element roughly represents a variation in mm.

```python
nipy.algorithms.registration.affine.rotation_mat2vec(R)
Rotation vector from rotation matrix R

Parameters  R : (3,3) array-like

Rotation matrix

Returns  vec : (3,) array

Rotation vector, where norm of vec is the angle theta, and the axis of rotation is given by vec / theta

```python
nipy.algorithms.registration.affine.rotation_vec2mat(r)
The rotation matrix is given by the Rodrigues formula:

R = Id + sin(theta)*Sn + (1-cos(theta))*Sn^2

with:

0 -nz ny

Sn = nz 0 -nx

-ny             nx 0

where n = r / ||r||
```
In case the angle $\|r\|$ is very small, the above formula may lead to numerical instabilities. We instead use a Taylor expansion around $\theta=0$:

$$R = I + \sin(\theta)/\theta \cdot S + (1-\cos(\theta))/\theta^2 \cdot S^2$$

leading to:

$$R = I + (1-\theta^2/6) \cdot S + (1/2-\theta^2/24) \cdot S^2$$

To avoid numerical instabilities, an upper threshold is applied to the angle. It is chosen to be a multiple of $2\pi$, hence the resulting rotation is then the identity matrix. This strategy warrants that the output matrix is a continuous function of the input vector.

```python
nipy.algorithms.registration.affine.slices2aff(slices)
```

Return affine from start, step of sequence `slices` of slice objects

- **Parameters** `slices`: sequence of slice objects
- **Returns** `aff`: ndarray

If $N = \text{len}(\text{slices})$ then affine is shape (N+1, N+1) with diagonal given by the step attribute of the slice objects (where None corresponds to 1), and the $:N$ elements in the last column are given by the start attribute of the slice objects

**Examples**

```python
>>> slices2aff([slice(None), slice(None)])
array([[ 1., 0., 0.],
[ 0., 1., 0.],
[ 0., 0., 1.]])

>>> slices2aff([slice(2, 3, 4), slice(3, 4, 5), slice(4, 5, 6)])
array([[ 4., 0., 0., 2.],
[ 0., 5., 0., 3.],
[ 0., 0., 6., 4.],
[ 0., 0., 0., 1.]])
```

```python
nipy.algorithms.registration.affine.subgrid_affine(affine, slices)
```

Return dot product of `affine` and affine resulting from `slices`

- **Parameters** `affine`: array-like
  - Affine to apply on right of affine resulting from `slices`
- `slices`: sequence of slice objects
  - Slices generating (N+1, N+1) affine from `slices2aff`, where $N = \text{len}(\text{slices})$
- **Returns** `aff`: ndarray
  - result of `np.dot(affine, slice_affine)` where `slice_affine` is affine resulting from `slices2aff(slices)`.

- **Raises** `ValueError`: if the `slice_affine` contains non-integer values

```python
nipy.algorithms.registration.affine.threshold(x, th)
```

```python
nipy.algorithms.registration.affine.to_matrix44(t)
```

t is a vector of affine transformation parameters with size at least 6.

- size $< 6 \implies$ error size $== 6 \implies$ t is interpreted as translation + rotation size $== 7 \implies$ t is interpreted as translation + rotation + isotropic scaling size $< 12 \implies$ error size $>= 12 \implies$ t is interpreted as translation + rotation + scaling + pre-rotation
39.1 Module: algorithms.registration.chain_transform

Inheritance diagram for nipy.algorithms.registration.chain_transform:

registration.chain_transform.ChainTransform

Chain transforms

39.2 ChainTransform

class nipy.algorithms.registration.chain_transform.ChainTransform(optimizable, pre=None, post=None)

    Bases: object

    Methods

    apply(pts)  Apply full transformation to points pts

    __init__(optimizable, pre=None, post=None)
    Create chain transform instance
    
    Parameters  optimizable : array or Transform
                Transform that we are optimizing. If this is an array, then assume it’s an affine matrix.
    pre : None or array or Transform, optional
         If not None, a transform that should be applied to points before applying the optimizable transform. If an array, then assume it’s an affine matrix.
post : None or Transform, optional

If not None, a transform that should be applied to points after applying any pre transform, and then the optimizable transform. If an array, assume it’s an affine matrix

apply (pts)
Apply full transformation to points pts
If there are N points, then pts will be N by 3

Parameters pts : array-like
array of points

Returns transformed_pts : array
N by 3 array of transformed points

param
get/set param
40.1 Module: `algorithms.registration.groupwise_registration`

Inheritance diagram for `nipy.algorithms.registration.groupwise_registration`:

```
registration.groupwise_registration.Image4d
```

```
registration.groupwise_registration.Realign4dAlgorithm
```

```
registration.groupwise_registration.Realign4d
```

```
registration.groupwise_registration.FmriRealign4d
```

```
registration.groupwise_registration.Realign4dAlgorithm
```

40.2 Classes

40.2.1 `FmriRealign4d`

```python
class nipy.algorithms.registration.groupwise_registration.FmriRealign4d(images,
slice_order,
interleaved=None,
tr=1.0,
tr_slices=None,
start=0.0,
time_interp=True,
affine_class=<class 'nipy.algorithms.registration.affine.Rigid'>,
slice_info=None):
```

```
Bases: nipy.algorithms.registration.groupwise_registration.Realign4d
```

Methods
Neuroimaging in Python Documentation, Release 0.3.0

**resample**([r, align_runs])

Return the resampled run number r as a 4d nipy-like image.

**__init__**(images, slice_order=None, interleaved=None, tr=1.0, tr_slices=None, start=0.0, time_interp=True, affine_class=<class 'nipy.algorithms.registration.affine.Rigid'>, slice_info=None)

Spatiotemporal realignment class for fMRI series.

**Parameters**

- **images**: image or list of images
  - Single or multiple input 4d images representing one or several fMRI runs.

- **tr**: float
  - Inter-scan repetition time, i.e. the time elapsed between two consecutive scans. The unit in which tr is given is arbitrary although it needs to be consistent with the tr_slices and start arguments.

- **tr_slices**: float
  - Inter-slice repetition time, same as tr for slices. If None, acquisition is assumed continuous and tr_slices is set to tr divided by the number of slices.

- **start**: float
  - Starting acquisition time respective to the implicit time origin.

- **slice_order**: str or array-like
  - If str, one of {'ascending', 'descending'}. If array-like, then the order in which the slices were collected in time. For instance, the following represents an ascending contiguous sequence:
    
    slice_order = [0, 1, 2, ...]

- **interleaved**: bool
  - Deprecated.
  - Whether slice acquisition order is interleaved. Ignored if slice_order is array-like.

  - If slice_order=='ascending' and interleaved==True, the assumed slice order is:
    
    [0, 2, 4, ..., 1, 3, 5, ...]

  - If slice_order=='descending' and interleaved==True, the assumed slice order is:
    
    [N-2, N-4, N-6]

  - Given that there exist other types of interleaved acquisitions depending on scanner settings and manufacturers, it is strongly recommended to input the slice_order as an array unless you are sure what you are doing.

- **slice_info**: None or tuple, optional
  - None, or a tuple with slice axis as the first element and direction as the second, for instance (2, 1). If None, then guess the slice axis, and direction, as the closest to the z axis, as estimated from the affine.

**estimate**([loops, between_loops, align_runs, ...])

284 Chapter 40. algorithms.registration.groupwise_registration
resample \((r=None, \text{align\_runs}=True)\)
Return the resampled run number \(r\) as a 4d nipy-like image. Returns all runs as a list of images if \(r==\) None.

### 40.2.2 Image4d

class nipy.algorithms.registration.groupwise_registration.Image4d(data, affine, tr, tr_slices=None, start=0.0, slice_order='ascending', interleaved=None, slice_info=None)

Bases: object

Class to represent a sequence of 3d scans (possibly acquired on a slice-by-slice basis).
Object remains empty until the data array is actually loaded in memory.

**Parameters**
- **data**: nd array or proxy (function that actually gets the array)

**Methods**

- `free_data()`
- `get_data()`
- `get_shape()`
- `scanner_time(z, t)`
  \(tv = \text{scanner\_time}(z, t)\)
- `z_to_slice(z)`
  Account for the fact that slices may be stored in reverse order wrt the scanner coordinate system convention (slice 0 == bottom of the head)

### 40.2.3 Realign4d

class nipy.algorithms.registration.groupwise_registration.Realign4d(images, affine_class=<class 'nipy.algorithms.registration.affine.Rigid'>, n, use_preproc_nipy=False)

Bases: object
Methods

\texttt{\texttt{estimate}}([\texttt{loops, between\_loops, align\_runs, ...}])
\texttt{\texttt{resample}}([\texttt{r, align\_runs}]) Return the resampled run number \texttt{r} as a 4d nipy-like image.

\texttt{\texttt{\_\_init\_\_}}(\texttt{images, affine\_class=\texttt{\_\_class 'nipy.algorithms.registration.affine.Rigid'}})
\texttt{\texttt{estimate}}(\texttt{loops=5, between\_loops=None, align\_runs=True, speedup=5, borders=(1, 1, 1), optimizer='neg', xtol=1e-05, ftol=1e-05, gtol=1e-05, stepsize=1e-06, maxiter=64, maxfun=None, refscan=0})
\texttt{\texttt{resample}}(\texttt{r=None, align\_runs=True})
\hspace{10pt}Return the resampled run number \texttt{r} as a 4d nipy-like image. Returns all runs as a list of images if \texttt{r} == \texttt{None}. 
40.2.4 Realign4dAlgorithm

class nipy.algorithms.registration.groupwise_registration.Realign4dAlgorithm:

    im4d, affine_class=<class 'nipy.algorithms.registration.affine.Rigid'>, transforms=None, time_interp=True, subsampling=(1, 1, 1), borders=(1, 1, 1), optimizer='ncg', optimize_template=True, xtol=1e-05, ftol=1e-05, gtol=1e-05, stepsize=1e-06, max_iter=64, max_fun=None, ref_scan=0)

    Bases: object

    Methods

    align_to_refscan() The motion_estimate method aligns scans with an online template so that spatial transforms
    estimate_instant_motion(t) Estimate motion parameters at a particular time.
    estimate_motion() Optimize motion parameters for the whole sequence.
    init_instant_motion(t) Pre-compute and cache some constants (at fixed time) for repeated computations of the alignment.
    resample(t) Resample a particular time frame on the (sub-sampled) working grid.
    resample_full_data() Resample a particular time frame on the full working grid.
    set_fmin(optimizer, stepsize, **kwargs) Return the minimization function
    set_transform(t, pc)
align_to_refscan()

The motion_estimate method aligns scans with an online template so that spatial transforms map some average head space to the scanner space. To conventionally redefine the head space as being aligned with some reference scan, we need to right compose each head_average-to-scanner transform with the refscan’s ‘to head_average’ transform.

estimate_instant_motion(t)

Estimate motion parameters at a particular time.

estimate_motion()

Optimize motion parameters for the whole sequence. All the time frames are initially resampled according to the current space/time transformation, the parameters of which are further optimized sequentially.

init_instant_motion(t)

Pre-compute and cache some constants (at fixed time) for repeated computations of the alignment energy. The idea is to decompose the average temporal variance via:

\[ V = \frac{n-1}{n} V^* + \frac{n-1}{n^2} (x-m^*)^2 \]

with \( x \) the considered volume at time \( t \), and \( m^* \) the mean of all resampled volumes but \( x \). Only the second term is variable when one volumes while the others are fixed. A similar decomposition is used for the global variance, so we end up with:

\[ \frac{V}{V_0} = \frac{[nV^* + (x-m^*)^2]}{[nV_0^* + (x-m_0^*)^2]} \]

resample(t)

Resample a particular time frame on the (sub-sampled) working grid. \( x,y,z,t \) are “head” grid coordinates \( X,Y,Z,T \) are “scanner” grid coordinates.

resample_full_data()

set_fmin(optimizer, stepsize, **kwargs)

Return the minimization function.

set_transform(t, pc)

40.3 Functions

nipy.algorithms.registration.groupwise_registration.adjust_subsampling(speedup, dims)
nipy.algorithms.registration.groupwise_registration.interp_slice_order(Z, slice_order)
nipy.algorithms.registration.groupwise_registration.make_grid(dims, subsampling=(1, 1, 1), borders=(0, 0, 0))
nipy.algorithms.registration.groupwise_registration.realign4d(runs,
    affine_class=<class 'nipy.algorithms.registration.affine.Rigid'>,
    time_interp=True,
    align_runs=True,
    loops=5,  between_loops=5,
    speedup=5, borders=(1, 1, 1),
    optimizer='ncg',
    xtol=1e-05,
    ftol=1e-05,
    gtol=1e-05,
    stepsize=1e-06,
    maxiter=64,
    maxfun=None,
    refscan=0)

Parameters  runs: list of Image4d objects

Returns  transforms: list

    nested list of rigid transformations

    transforms map an ‘ideal’ 4d grid (conventionally aligned with the :
    first scan of the first run) to the ‘acquisition’ 4d grid for each :

        run :

nipy.algorithms.registration.groupwise_registration.resample4d(im4d,  transforms,
    time_interp=True)

Resample a 4D image according to the specified sequence of spatial transforms, using either 4D interpolation if

time_interp is True and 3D interpolation otherwise.

nipy.algorithms.registration.groupwise_registration.scanner_coords(xyz, affine,  from_world,
    to_world)
Realign a single run in space and time.

**Parameters**

`im4d` : Image4d instance

`speedup` : int or sequence

If a sequence, implement a multi-scale
41.1 Module: algorithms.registration.histogram_registration

Intensity-based image registration

41.2 Class

41.3 HistogramRegistration

class nipy.algorithms.registration.histogram_registration.HistogramRegistration (from_img, to_img, from_bins=256, to_bins=None, from_mask=None, to_mask=None, similarity='crl1', interp='pv', **kwargs)

Bases: object

A class to represent a generic intensity-based image registration algorithm.
Methods

**eval**(T)  
Evaluate similarity function given a world-to-world transform.

**explore**(T0, *args)  
Evaluate the similarity at the transformations specified by sequences of parameter values.

**optimize**(T[, optimizer])  
Optimize transform T with respect to similarity measure.

**set_fov**(spacing, corner, size, npoints)  
Defines a subset of the from image to restrict joint histogram computation.

**subsample**(spacing, npoints)  

__init__ (from_img, to_img, from_bins=256, to_bins=None, from_mask=None, to_mask=None, similarity='crl1', interp='pv', **kwargs)  
Creates a new histogram registration object.

**Parameters**  
from_img : nipy-like image
  From image

  to_img  [nipy-like image]  To image

  from_bins  [integer]  Number of histogram bins to represent the from image

  to_bins  [integer]  Number of histogram bins to represent the to image

  from_mask  [array-like]  Mask to apply to the from image

  to_mask  [array-like]  Mask to apply to the to image

  similarity  [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.

  interp  : str

**eval**(T)  
Evaluate similarity function given a world-to-world transform.

  **Parameters**  
  T : Transform

  Transform object implementing apply method

**explore**(T0, *args)  
Evaluate the similarity at the transformations specified by sequences of parameter values.

  For instance:

  explore(T0, (0, [-1,0,1]), (4, [-2.,2]))

**interp**

**optimize**(T, optimizer='powell', **kwargs)  
Optimize transform T with respect to similarity measure.

  The input object T will change as a result of the optimization.

  **Parameters**  
  T : object or str
An object representing a transformation that should implement apply method and param attribute or property. If a string, one of ‘rigid’, ‘similarity’, or ‘affine’. The corresponding transformation class is then initialized by default.

**optimizer** : str
Name of optimization function (one of ‘powell’, ‘steepest’, ‘cg’, ‘bfgs’, ‘simplex’)

**kwargs** : dict
keyword arguments to pass to optimizer

**set_fov** *(spacing=None, corner=(0, 0, 0), size=None, npoints=None)*
Defines a subset of the from image to restrict joint histogram computation.

**Parameters**
- **spacing** : sequence (3,) of positive integers
  Subsampling of image in voxels, where None (default) results in the subsampling to be automatically adjusted to roughly match a cubic grid with npoints voxels
- **corner** : sequence (3,) of positive integers
  Bounding box origin in voxel coordinates
- **size** : sequence (3,) of positive integers
  Desired bounding box size
- **npoints** : positive integer
  Desired number of voxels in the bounding box. If a spacing argument is provided, then npoints is ignored.

**similarity**

**subsample** *(spacing=None, npoints=None)*

### 41.4 Functions

**nipy.algorithms.registration.histogram_registration.clamp**(x, bins=256, mask=None)
Clamp array values that fall within a given mask in the range [0..bins-1] and reset masked values to -1.

**Parameters**
- **x** : ndarray
  The input array
- **bins** : number
  Desired number of bins
- **mask** : ndarray, tuple or slice
  Anything such that x[mask] is an array.

**Returns**
- **y** : ndarray
  Clamped array, masked items are assigned -1
- **bins** : number
  Adjusted number of bins

**nipy.algorithms.registration.histogram_registration.ideal_spacing**(data, npoints)
Tune spacing factors so that the number of voxels in the output block matches a given number.
Parameters

**data**: ndarray or sequence

Data image to subsample

**npoints**: number

Target number of voxels (negative values will be ignored)

Returns **spacing**: ndarray:

Spacing factors

```python
nipy.algorithms.registration.histogram_registration.smallest_bounding_box(msk)
```

Extract the smallest bounding box from a mask
ALGORITHMS.REGISTRATION.OPTIMIZER

42.1 Module: algorithms.registration.optimizer

42.2 Functions

nipy.algorithms.registration.optimizer.configure_optimizer(optimizer, fprime=None, fhess=None, **kwargs)

Return the minimization function

nipy.algorithms.registration.optimizer.subdict(dic, keys)

nipy.algorithms.registration.optimizer.use_derivatives(optimizer)
43.1 Module: algorithms.registration.polyaffine

Inheritance diagram for nipy.algorithms.registration.polyaffine:

```
registration.transform.Transform --|-- registration.polyaffine.PolyAffine
```

43.2 PolyAffine

class nipy.algorithms.registration.polyaffine.PolyAffine(centers, affines, sigma, glob_affine=None)

**Bases:** nipy.algorithms.registration.transform.Transform

**Methods**

- `affine(i)`
- `affines()`
- `apply(xyz)` xyz is an (N, 3) array
- `compose(other)` Compose this transform onto another
- `left_compose(other)`

```python
__init__(centers, affines, sigma, glob_affine=None)
```

centers: N times 3 array

We are given a set of affine transforms $T_i$ with centers $x_i$, all in homogeneous coordinates. The polyaffine transform is defined, up to a right composition with a global affine, as:

$$T(x) = \sum_i w_i(x) T_i x$$

where $w_i(x) = g(x-x_i)/Z(x)$ are normalized Gaussian weights that sum up to one for every $x$. 
affine

affines()

apply(apply)

xyz is an (N, 3) array

compose(compose)

Compose this transform onto another

Parameters other: Transform

transform that we compose onto

Returns composed_transform: Transform

a transform implementing the composition of self on other

left_compose(left_compose)

param
44.1 Module: algorithms.registration.resample

```python
def resample(moving, transform=None, reference=None, mov_voxel_coords=False, ref_voxel_coords=False, dtype=None, interp_order=3):
    # Function to resample moving into voxel space of reference using transform

    # Apply a transformation to the image considered as 'moving' to bring it into the same grid as a given reference image. The transformation usually maps world space in reference to world space in moving, but can also be a voxel to voxel mapping (see parameters below).

    # This function uses scipy.ndimage except for the case interp_order==3, where a fast cubic spline implementation is used.

    Parameters
    ----------
    moving : nipy-like image
        Image to be resampled.
    transform : transform object or None
        Represents a transform that goes from the reference image to the moving image. None means an identity transform. Otherwise, it should have either an apply method, or an as_affine method. By default, transform maps between the output (world) space of reference and the output (world) space of moving. If mov_voxel_coords is True, maps to the voxel space of moving and if ref_voxel_coords is True, maps from the voxel space of reference.
    reference : None or nipy-like image or tuple, optional
        The reference image defines the image dimensions and xyz affine to which to resample. It can be input as a nipy-like image or as a tuple (shape, affine). If None, use moving to define these.
    mov_voxel_coords : boolean, optional
        True if the transform maps to voxel coordinates, False if it maps to world coordinates.
    ref_voxel_coords : boolean, optional
        True if the transform maps from voxel coordinates, False if it maps from world coordinates.
    interp_order : int, optional
        Spline interpolation order, defaults to 3.
```

299
Returns

aligned_img : Image

Image resliced to reference with reference-to-moving transform transform
45.1 Module: `algorithms.registration.similarity_measures`

Inheritance diagram for `nipy.algorithms.registration.similarity_measures`:

45.2 Classes

45.2.1 `CorrelationCoefficient`

```python
class nipy.algorithms.registration.similarity_measures.CorrelationCoefficient(shape, **kwargs):
    Bases: nipy.algorithms.registration.similarity_measures.SimilarityMeasure
```

Methods
__call__(H)
averaged_loss(H)
loss(H)
npoints(H)
overall_loss(H)

__init__(shape, **kwargs)

averaged_loss (H)
loss (H)
npoints (H)
overall_loss (H)

45.2.2 CorrelationRatio

class nipy.algorithms.registration.similarity_measures.CorrelationRatio(shape, **kwargs)
Bases: nipy.algorithms.registration.similarity_measures.SimilarityMeasure

Methods

__call__(H)
averaged_loss(H)
loss(H)
npoints(H)
overall_loss(H)

__init__(shape, **kwargs)

averaged_loss (H)
loss (H)
npoints (H)
overall_loss (H)

45.2.3 CorrelationRatioL1

class nipy.algorithms.registration.similarity_measures.CorrelationRatioL1(shape, **kwargs)
Bases: nipy.algorithms.registration.similarity_measures.SimilarityMeasure

Methods

__call__(H)
averaged_loss(H)

Continued on next page
Table 45.3 – continued from previous page

<table>
<thead>
<tr>
<th>loss(H)</th>
</tr>
</thead>
<tbody>
<tr>
<td>npoints(H)</td>
</tr>
<tr>
<td>overall_loss(H)</td>
</tr>
</tbody>
</table>

```python
__init__(shape, **kwargs)

averaged_loss(H)
l(H)
npoints(H)
overall_loss(H)
```

### 45.2.4 DiscreteParzenMutualInformation

class nipy.algorithms.registration.similarity_measures.DiscreteParzenMutualInformation(shape, **kwargs)

Bases: nipy.algorithms.registration.similarity_measures.SimilarityMeasure

Methods

```python
__call__(H)

averaged_loss(H)
l(H)
npoints(H)
overall_loss(H)
```

```python
__init__(shape, **kwargs)

averaged_loss(H)
l(H)
npoints(H)
overall_loss(H)
```

### 45.2.5 MutualInformation

class nipy.algorithms.registration.similarity_measures.MutualInformation(shape, **kwargs)

Bases: nipy.algorithms.registration.similarity_measures.SimilarityMeasure

Methods

```python
__call__(H)

averaged_loss(H)
l(H)
npoints(H)
```

Continued on next page
45.2.6 NormalizedMutualInformation

class nipy.algorithms.registration.similarity_measures.NormalizedMutualInformation(shape, **kwargs)

Bases: nipy.algorithms.registration.similarity_measures.SimilarityMeasure

NMI = 2*(1 - H(I,J)/[H(I)+H(J)]) = 2*MI/[H(I)+H(J)]

Methods

__call__(H)
averaged_loss(H)
loss(H)
npoints(H)
overall_loss(H)

45.2.7 ParzenMutualInformation

class nipy.algorithms.registration.similarity_measures.ParzenMutualInformation(shape, **kwargs)

Bases: nipy.algorithms.registration.similarity_measures.SimilarityMeasure

Methods

__call__(H)
averaged_loss(H)
loss(H)
npoints(H)
overall_loss(H)
__init__ (shape, **kwargs)

averaged_loss (H)

loss (H)

npoints (H)

overall_loss (H)

45.2.8 SimilarityMeasure

class nipy.algorithms.registration.similarity_measures.SimilarityMeasure (shape, **kwargs)

Bases: object

Methods

__call__ (H)

averaged_loss (H)

loss (H)

npoints (H)

overall_loss (H)

__init__ (shape, **kwargs)

averaged_loss (H)

loss (H)

npoints (H)

overall_loss (H)

45.2.9 SupervisedLikelihoodRatio

class nipy.algorithms.registration.similarity_measures.SupervisedLikelihoodRatio (shape, **kwargs)

Bases: nipy.algorithms.registration.similarity_measures.SimilarityMeasure

Methods

__call__ (H)

averaged_loss (H)

loss (H)

npoints (H)

overall_loss (H)

__init__ (shape, **kwargs)

averaged_loss (H)

loss (H)
45.3 Functions

\begin{verbatim}
npy.algorithms.registration.similarity_measures.dist2loss(dist, margI=None, margJ=None)
npy.algorithms.registration.similarity_measures.nonzero(x)
\end{verbatim}

Force strictly positive values.
ALGORITHMS.REGISTRATION.TRANSFORM

46.1 Module: algorithms.registration.transform

Inheritance diagram for nipy.algorithms.registration.transform:

registration.transform.Transform

Generic transform class

This implementation specifies an API. We’ve done our best to avoid checking instances, so any class implementing this API should be valid in the places (like registration routines) that use transforms. If that isn’t true, it’s a bug.

46.2 Transform

class nipy.algorithms.registration.transform.Transform(func)
    Bases: object

    A default transformation class

    This class specifies the tiny API. That is, the class should implement:

    • obj.param - the transformation exposed as a set of parameters. Changing param should change the transformation
    • obj.apply(pts) - accepts (N,3) array-like of points in 3 dimensions, returns an (N, 3) array of transformed points
    • obj.compose(xform) - accepts another object implementing apply, and returns a new transformation object, where the resulting transformation is the composition of the obj transform onto the xform transform.

    Methods
apply(pts)
compose(other)

__init__(func)
apply(pts)
compose(other)
param
47.1 Module: algorithms.resample

Some simple examples and utility functions for resampling.

47.2 Functions

nipy.algorithms.resample.resample(image, target, mapping, shape, order=3)

Resample image to target CoordinateMap

Use a “world-to-world” mapping mapping and spline interpolation of a order.

Here, “world-to-world” refers to the fact that mapping should be a callable that takes a physical coordinate in “target” and gives a physical coordinate in “image”.

Parameters

**image**: Image instance

image that is to be resampled

**target**: CoordinateMap

coordinate map for output image

**mapping**: callable or tuple or array

transformation from target.function_range to image.coordmap.function_range, i.e. ‘world-to-world mapping’. Can be specified in three ways: a callable, a tuple (A, b) representing the mapping y=dot(A,x)+b or a representation of this mapping as an affine array, in homogeneous coordinates.

**shape**: sequence of int

shape of output array, in target.function_domain

**order**: int, optional

what order of interpolation to use in scipy.ndimage

Returns

**output**: Image instance

with interpolated data and output.coordmap == target

nipy.algorithms.resample.resample_img2img(source, target, order=3)

Resample source image to space of target image
This wraps the resample function to resample one image onto another. The output of the function will give an image with shape of the target and data from the source.

**Parameters**

- **source**: Image
  - Image instance that is to be resampled
- **target**: Image
  - Image instance to which source is resampled. The output image will have the same shape as the target, and the same coordmap
- **order**: int, optional
  - What order of interpolation to use in `scipy.ndimage`

**Returns**

- **output**: Image
  - Image with interpolated data and output.coordmap == target.coordmap

**Examples**

```python
generate code
```
48.1 Module: algorithms.segmentation.brain_segmentation

Inheritance diagram for nipy.algorithms.segmentation.brain_segmentation:

```python
nipy.algorithms.segmentation.brain_segmentation.BrainT1Segmentation
```

48.2 BrainT1Segmentation

```python
class nipy.algorithms.segmentation.brain_segmentation.BrainT1Segmentation:

data, mask=None, model='3k', niter=25, ngb_size=6, beta=0.5, ref_params=None, init_params=None, convert=True)
```

Methods

```python
convert()
__init__(data, mask=None, model='3k', niter=25, ngb_size=6, beta=0.5, ref_params=None, init_params=None, convert=True)
convert()```
49.1 Module: `algorithms.segmentation.segmentation`

Inheritance diagram for `nipy.algorithms.segmentation.segmentation`:

```
segmentation.segmentation.Segmentation
```

49.2 Class

49.3 Segmentation

class `nipy.algorithms.segmentation.segmentation.Segmentation`

```
(data, mask=None, mu=None, sigma=None, ppm=None, prior=None, U=None, ngb_size=26, beta=0.1)
```

Bases: `object`

Methods

```
free_energy([ppm])          # Compute the free energy defined as:
log_external_field()        # Compute the logarithm of the external field, where the
map()                       # Return the maximum a posterior label map
normalized_external_field() #
run(niters, freeze)         #
set_markov_prior(beta, U)   #
```

Continued on next page
Table 49.1 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ve_step()</td>
<td></td>
</tr>
<tr>
<td>vm_step([freeze])</td>
<td></td>
</tr>
</tbody>
</table>

__init__(data, mask=None, mu=None, sigma=None, ppm=None, prior=None, U=None, ngh_size=26, beta=0.1)

Class for multichannel Markov random field image segmentation using the variational EM algorithm. For details regarding the underlying algorithm, see:


**Parameters**
- **data**: array-like
  - Input image array
- **mask**: array-like or tuple of array
  - Input mask to restrict the segmentation
- **beta**: float
  - Markov regularization parameter
- **mu**: array-like
  - Initial class-specific means
- **sigma**: array-like
  - Initial class-specific variances

**free_energy(ppm=None)**

Compute the free energy defined as:

\[ F(q, \theta) = \int q(x) \log \frac{q(x)}{p(x,y/\theta)} \, dx \]

associated with input parameters mu, sigma and beta (up to an ignored constant).

**log_external_field()**

Compute the logarithm of the external field, where the external field is defined as the likelihood times the first-order component of the prior.

**map()**

Return the maximum a posterior label map

**normalized_external_field()**

**run(niters=10, freeze=())**

**set_markov_prior(beta, U=None)**

**ve_step()**

**vm_step(freeze=())**

49.4 Functions

nipy.algorithms.segmentation.segmentation.binarize_ppm(q)

Assume input ppm is masked (ndim==2)

nipy.algorithms.segmentation.segmentation.map_from_ppm(ppm, mask=None)
nipy.algorithms.segmentation.segmentation.moment_matching(dat, mu, sigma, glob_mu, glob_sigma)

Moment matching strategy for parameter initialization to feed a segmentation algorithm.

**Parameters**

- **data**: array
  - Image data.
- **mu**: array
  - Template class-specific intensity means
- **sigma**: array
  - Template class-specific intensity variances
- **glob_mu**: float
  - Template global intensity mean
- **glob_sigma**: float
  - Template global intensity variance

**Returns**

- **dat_mu**: array
  - Guess of class-specific intensity means
- **dat_sigma**: array
  - Guess of class-specific intensity variances
ALGORITHMS.STATISTICS.BENCH.BENCH_INTVOL

50.1 Module: algorithms.statistics.bench.bench_intvol

50.2 Functions

nipy.algorithms.statistics.bench.bench_intvol.bench_lips1d()
nipy.algorithms.statistics.bench.bench_intvol.bench_lips2d()
nipy.algorithms.statistics.bench.bench_intvol.bench_lips3d()
51.1 Module: algorithms.statistics.empirical_pvalue

Inheritance diagram for nipy.algorithms.statistics.empirical_pvalue:

```
statistics.empirical_pvalue.NormalEmpiricalNull
```

Routines to get corrected p-values estimates, based on the observations.
It implements 3 approaches:

- a class that fits a Gaussian model to the central part of an histogram, following [1]


  This is typically necessary to estimate a FDR when one is not certain that the data behaves as a standard normal under $H_0$.

- a model based on Gaussian mixture modelling 'a la Oxford'

Author: Bertrand Thirion, Yaroslav Halchenko, 2008-2012

51.2 Class

51.3 NormalEmpiricalNull

```
class nipy.algorithms.statistics.empirical_pvalue.NormalEmpiricalNull(x):
    Bases: object
```

Class to compute the empirical null normal fit to the data.
The data which is used to estimate the FDR, assuming a Gaussian null from Schwartzmann et al., NeuroImage 44 (2009) 71–82

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fdr(theta)</code></td>
<td>Given a threshold theta, find the estimated FDR</td>
</tr>
<tr>
<td><code>fdrcurve()</code></td>
<td>Returns the FDR associated with any point of self.x</td>
</tr>
<tr>
<td><code>learn([left, right])</code></td>
<td>Estimate the proportion, mean and variance of a Gaussian distribution</td>
</tr>
<tr>
<td><code>plot([efp, alpha, bar, mpaxes])</code></td>
<td>Plot the histogram of x</td>
</tr>
<tr>
<td><code>threshold([alpha, verbose])</code></td>
<td>Compute the threshold corresponding to an alpha-level FDR for x</td>
</tr>
<tr>
<td><code>uncorrected_threshold([alpha, verbose])</code></td>
<td>Compute the threshold corresponding to a specificity alpha for x</td>
</tr>
</tbody>
</table>

**__init__(x)__**

Initialize an empirical null normal object.

**Parameters**

- `x` : 1D ndarray
  
  The data used to estimate the empirical null.

**Methods**

**fdr(theta)**

Given a threshold theta, find the estimated FDR

**Parameters**

- `theta` : float or array of shape (n_samples)
  
  Values to test

**Returns**

- `afp` : value of array of shape(n)

**fdrcurve**

Returns the FDR associated with any point of self.x

**learn([left, right])**

Estimate the proportion, mean and variance of a Gaussian distribution for a fraction of the data

**Parameters**

- `left` : float, optional
  
  Left cut parameter to prevent fitting non-gaussian data

- `right` : float, optional
  
  Right cut parameter to prevent fitting non-gaussian data

**Notes**

This method stores the following attributes:

- `mu = mu`
- `p0 = min(1, np.exp(lp0))`
- `sqsigma`: standard deviation of the estimated normal distribution
- `sigma`: `np.sqrt(sqsigma)`: variance of the estimated normal distribution

**plot([efp=None, alpha=0.05, bar=1, mpaxes=None])**

Plot the histogram of x

**Parameters**

- `efp` : float, optional
Neuroimaging in Python Documentation, Release 0.3.0

The empirical FDR (corresponding to x) if efp==None, the false positive rate threshold plot is not drawn.

**alpha**: float, optional
The chosen FDR threshold

**bar**: bool, optional

**mpaxes=None**: if not None, handle to an axes where the fig will be drawn. Avoids creating unnecessarily new figures:

**threshold (alpha=0.05, verbose=0)**
Compute the threshold corresponding to an alpha-level FDR for x

**Parameters**
- **alpha**: float, optional
  - the chosen false discovery rate threshold.
- **verbose**: boolean, optional
  - the verbosity level, if True a plot is generated.

**Returns**
- **theta**: float:
  - the critical value associated with the provided FDR

**uncorrected_threshold (alpha=0.001, verbose=0)**
Compute the threshold corresponding to a specificity alpha for x

**Parameters**
- **alpha**: float, optional
  - the chosen false discovery rate (FDR) threshold.
- **verbose**: boolean, optional
  - the verbosity level, if True a plot is generated.

**Returns**
- **theta**: float:
  - the critical value associated with the provided p-value

## 51.4 Functions

**nipy.algorithms.statistics.empirical_pvalue.check_p_values (p_values)**
Basic checks on the p_values array: values should be within [0,1]
Assures also that p_values are at least in 1d array. None of the checks is performed if p_values is None.

**Parameters**
- **p_values**: array of shape (n)
  - The sample p-values

**Returns**
- **p_values**: array of shape (n)
  - The sample p-values

**nipy.algorithms.statistics.empirical_pvalue.fdr (p_values=None, verbose=0)**
Returns the FDR associated with each p value

**Parameters**
- **p_values**: ndarray of shape (n)
  - The samples p-value

**Returns**
- **q**: array of shape(n)
The corresponding fdr values

```python
nipy.algorithms.statistics.empirical_pvalue.fdr_threshold(p_values, alpha=0.05)
```

Return FDR threshold given p values

**Parameters**
- **p_values**: array of shape (n), optional
  - The samples p-value
- **alpha**: float, optional
  - The desired FDR significance

**Returns**
- **critical_p_value**: float
  - The p value corresponding to the FDR alpha

```python
nipy.algorithms.statistics.empirical_pvalue.gamma_gaussian_fit(x, test=None, verbose=0, mpaxes=None, bias=1, gaussian_mixture=0, return_estimator=False)
```

Computing some prior probabilities that the voxels of a certain map are in class disactivated, null or active using a gamma-Gaussian mixture

**Parameters**
- **x**: array of shape (nvox,)
  - the map to be analysed
- **test**: array of shape (nbitems,), optional
  - the test values for which the p-value needs to be computed by default, test = x
- **verbose**: 0, 1 or 2, optional
  - verbosity mode, 0 is quiet, and 2 calls matplotlib to display graphs.
- **mpaxes**: matplotlib axes, optional
  - axes handle used to plot the figure in verbose mode if None, new axes are created
- **bias**: float, optional
  - lower bound on the Gaussian variance (to avoid shrinkage)
- **gaussian_mixture**: float, optional
  - if nonzero, lower bound on the Gaussian mixing weight (to avoid shrinkage)
- **return_estimator**: boolean, optional
  - if return_estimator is true, the estimator object is returned.

**Returns**
- **bfp**: array of shape (nbitems,3)
  - The probability of each component in the mixture model for each test value
- **estimator**: nipy.labs.clustering.gmmixture.GGGM object
  - The estimator object, returned only if return_estimator is true.

```python
nipy.algorithms.statistics.empirical_pvalue.gaussian_fdr(x)
```

Return the FDR associated with each value assuming a Gaussian distribution

```python
nipy.algorithms.statistics.empirical_pvalue.gaussian_fdr_threshold(x, alpha=0.05)
```

Return FDR threshold given normal variates
Given an array $x$ of normal variates, this function returns the critical p-value associated with $\alpha$. $x$ is explicitly assumed to be normal distributed under $H_0$.

**Parameters**

$x$: ndarray
- input data

$\alpha$: float, optional
- desired significance

**Returns**

$\text{threshold}$: float
- threshold, given as a Gaussian critical value

```python
nipy.algorithms.statistics.empirical_pvalue.smoothed_histogram_from_samples(x, bins=None, nbins=256, normalized=False)
```

Returns the smooth histogram corresponding to the density underlying the samples in $x$.

**Parameters**

$x$: array of shape(n_samples),
- input data

$\text{bins}$: array of shape(nbins+1), optional,
- the bins location

$\text{nbins}$: int, optional,
- the number of bins of the resulting histogram

$\text{normalized}$: bool, optional
- if True, the result is returned as a density value

**Returns**

$h$: array of shape (nbins):
- the histogram

$\text{bins}$: array of shape(nbins+1),
- the bins location

```python
nipy.algorithms.statistics.empirical_pvalue.three_classes_GMM_fit(x, test=None, alpha=0.01, prior_strength=100, verbose=0, fixed_scale=False, mpaxes=None, bias=0, theta=0, return_estimator=False)
```

Fit the data with a 3-classes Gaussian Mixture Model, i.e. compute some probability that the voxels of a certain map are in class deactivated, null or active.

**Parameters**

$x$: array of shape (nvox,1),
- The map to be analysed

$\text{test}$: array of shape(nbitems,1), optional,
the test values for which the p-value needs to be computed by default (if None), test=x

**alpha**: float, optional:
the prior weights of the positive and negative classes

**prior_strength**: float, optional:
the confidence on the prior (should be compared to size(x))

**verbose**: int:
verbosity mode

**fixed_scale**: bool, optional:
boolean, variance parameterization. if True, the variance is locked to 1 otherwise, it is estimated from the data

**mpaxes**::
axes handle used to plot the figure in verbose mode if None, new axes are created

**bias**: bool:
allows a rescaling of the posterior probability that takes into account the threshold theta.
Not rigorous.

**theta**: float:
the threshold used to correct the posterior p-values when bias=1; normally, it is such that test>theta note that if theta = -np.inf, the method has a standard behaviour

**return_estimator**: boolean, optional:
If return_estimator is true, the estimator object is returned.

**Returns**

**bfp**: array of shape (nbitems,3):
the posterior probability of each test item belonging to each component in the GMM (sum to 1 across the 3 classes) if np.size(test)==0, i.e. nbitem==0, None is returned

**estimator**: nipy.labs.clustering.GMM object
The estimator object, returned only if return_estimator is true.

**Notes**

Our convention is that:

- class 1 represents the negative class
- class 2 represents the null class
- class 3 represents the positive class
52.1 Module: `algorithms.statistics.formula.formulae`

Inheritance diagram for `nipy.algorithms.statistics.formula.formulae`:

52.1.1 Formula objects

A formula is basically a sympy expression for the mean of something of the form:

\[
\text{mean} = \sum ([\text{Beta}(e) \times e \text{ for } e \text{ in } \text{expr}])
\]

Or, a linear combination of sympy expressions, with each one multiplied by its own “Beta”. The elements of expr can be instances of Term (for a linear regression formula, they would all be instances of Term). But, in general, there might be some other parameters (i.e. sympy.Symbol instances) that are not Terms.

The design matrix is made up of columns that are the derivatives of mean with respect to everything that is not a Term, evaluated at a recarray that has field names given by `[str(t) for t in self.terms]`.

For those familiar with R’s formula syntax, if we wanted a design matrix like the following:

```r
> s.table = read.table("http://www-stat.stanford.edu/~jtaylo/courses/stats191/data/supervisor.table", header=T)
> d = model.matrix(lm(Y ~ X1*X3, s.table))
> d
   (Intercept) X1 X3 X1:X3
   1      51   39  1989
   2      64   54  3456
   3      70   69  4830
   4      63   47  2961
   5      78   66  5148
```
With the Formula, it looks like this:

```python
>>> r = np.rec.array([...
... (43, 51, 30, 39, 61, 92, 45), (63, 64, 51, 54, 63, 73, 47),...
... (71, 70, 68, 69, 76, 86, 48), (61, 63, 45, 47, 54, 84, 35),
... (81, 78, 56, 66, 71, 83, 47), (43, 55, 49, 44, 54, 49, 34),
... (58, 67, 42, 56, 66, 68, 35), (71, 75, 50, 55, 70, 66, 41),
... (72, 82, 72, 67, 71, 83, 31), (67, 61, 45, 47, 62, 80, 41),
... (64, 53, 58, 58, 67, 34), (67, 60, 47, 39, 59, 74, 41),
... (69, 52, 57, 42, 55, 63, 25), (68, 83, 83, 45, 59, 77, 35),
... (77, 75, 54, 72, 79, 77, 46), (81, 90, 50, 72, 60, 54, 36),
... (74, 85, 64, 69, 79, 79, 63), (65, 60, 65, 75, 55, 80, 60),
... (65, 70, 46, 57, 75, 85, 46), (50, 58, 68, 54, 64, 78, 52),
... (50, 40, 33, 34, 43, 64, 33), (64, 61, 52, 62, 66, 80, 41),
... (53, 66, 52, 50, 63, 80, 37), (40, 37, 42, 58, 50, 57, 49),
... (63, 54, 42, 48, 66, 75, 33), (66, 77, 66, 63, 88, 76, 72),
... (78, 75, 58, 74, 80, 78, 49), (48, 57, 44, 45, 51, 83, 38),
... (85, 85, 71, 71, 77, 74, 55), (82, 82, 39, 59, 64, 78, 39)],
... dtype=[('y', '<i8'), ('x1', '<i8'), ('x3', '<i8',), ('x4', '<i8'), ('x5', '<i8'),
... ('x6', '<i8'))]
>>> x1 = Term('x1'); x3 = Term('x3')
>>> f = Formula([x1, x3, x1*x3]) + I
>>> f.mean
_b0*x1 + _b1*x3 + _b2*x1*x3 + _b3
```

The I is the “intercept” term, I have explicitly not used R’s default of adding it to everything.

```python
>>> f.design(r)
array([[51.0, 39.0, 1989.0, 1.0), (64.0, 54.0, 3456.0, 1.0),
... (61.0, 57.0, 1843.0, 1.0), (65.0, 58.0, 3732.0, 1.0),
... (69.0, 52.0, 1744.0, 1.0), (77.0, 74.0, 5294.0, 1.0),
... (74.0, 85.0, 64.0, 69.0, 79.0, 79.0, 63.0), (65.0, 60.0, 65.0, 75.0, 55.0, 80.0, 60.0),
... (65.0, 70.0, 46.0, 57.0, 75.0, 85.0, 46.0), (50.0, 58.0, 68.0, 54.0, 64.0, 78.0, 52.0),
... (50.0, 40.0, 33.0, 34.0, 43.0, 64.0, 33.0), (64.0, 61.0, 52.0, 62.0, 66.0, 80.0, 41.0),
... (53.0, 66.0, 52.0, 50.0, 63.0, 80.0, 37.0), (40.0, 37.0, 42.0, 58.0, 50.0, 57.0, 49.0),
... (63.0, 54.0, 42.0, 48.0, 66.0, 75.0, 33.0), (66.0, 77.0, 66.0, 63.0, 88.0, 76.0, 72.0),
... (78.0, 75.0, 58.0, 74.0, 80.0, 78.0, 49.0), (48.0, 57.0, 44.0, 45.0, 51.0, 83.0, 38.0),
... (85.0, 85.0, 71.0, 71.0, 77.0, 74.0, 55.0), (82.0, 82.0, 39.0, 59.0, 64.0, 78.0, 39.0)]},
... dtype=[('y', '<i8'), ('x1', '<i8'), ('x3', '<i8'), ('x4', '<i8'), ('x5', '<i8'),
... ('x6', '<i8'))]
```
52.2 Classes

52.2.1 Beta

class nipy.algorithms.statistics.formula.formulae.Beta
  Bases: sympy.core.symbol.Dummy

A symbol tied to a Term term

Methods

__call__(*args)
adjoint()
apart([x])  See the apart function in sympy.polys
args_cnc([cset, warn])  Return [commutative factors, non-commutative factors] of self.
as_base_exp()
as_coeff_Add()  Efficiently extract the coefficient of a summation.
as_coeff_Mul([rational])  Efficiently extract the coefficient of a product.
as_coeff_add(*deps)  Return the tuple (c, args) where self is written as an Add, a.
as_coeff_exponent(x)  c*x**e -> c, e where x can be any symbolic expression.
as_coeff_factors(*deps)  This method is deprecated.
as_coeff_mul(*deps)  Return the tuple (c, args) where self is written as a Mul, m.
as_coeff_terms(*deps)  This method is deprecated.
as_coefficient(expr)  Extracts symbolic coefficient at the given expression.
as_coefficients_dict()  Return a dictionary mapping terms to their Rational coefficient.
as_content_primitive([radical])  This method should recursively remove a Rational from all arguments
as_dummy()
as_expr(*gens)  Convert a polynomial to a SymPy expression.
as_independent(*deps, **hint)  A mostly naive separation of a Mul or Add into arguments that are not
as_leading_term(*args, **kw_args)  Returns the leading (nonzero) term of the series expansion of self.
as_numer_denom()  expression -> a/b -> a, b
as_ordered_factors([order])  Return list of ordered factors (if Mul) else [self].
as_ordered_terms([order, data])  Transform an expression to an ordered list of terms.
as_poly(*gens, **args)  Converts self to a polynomial or returns None.
Table 52.1 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>as_powers_dict()</code></td>
<td>Return self as a dictionary of factors with each factor being treated as a power.</td>
</tr>
<tr>
<td><code>as_real_imag([deep])</code></td>
<td>Transform an expression to a list of terms.</td>
</tr>
<tr>
<td><code>as_terms()</code></td>
<td>Returns the atoms that form the current object.</td>
</tr>
<tr>
<td><code>cancel(*gens, **args)</code></td>
<td>See the cancel function in sympy.polys</td>
</tr>
<tr>
<td><code>class_key()</code></td>
<td></td>
</tr>
<tr>
<td><code>coeff(x[, n, right])</code></td>
<td>Returns the coefficient from the term containing “x**n” or None if</td>
</tr>
<tr>
<td><code>collect(syms[, func, evaluate, exact, ...])</code></td>
<td>See the collect function in sympy.simplify</td>
</tr>
<tr>
<td><code>combsimp()</code></td>
<td>See the combsimp function in sympy.simplify</td>
</tr>
<tr>
<td><code>compare(other)</code></td>
<td>Return -1, 0, 1 if the object is smaller, equal, or greater than other.</td>
</tr>
<tr>
<td><code>compare_pretty(*args, **kwargs)</code></td>
<td>Is a &gt; b in the sense of ordering in printing?</td>
</tr>
<tr>
<td><code>compute_leading_term(x[, skip_abs, logx])</code></td>
<td>as_leading_term is only allowed for results of .series()</td>
</tr>
<tr>
<td><code>conjugate()</code></td>
<td></td>
</tr>
<tr>
<td><code>copy()</code></td>
<td></td>
</tr>
<tr>
<td><code>could_extract_minus_sign()</code></td>
<td>Canonical way to choose an element in the set {e, -e} where e is any expression.</td>
</tr>
<tr>
<td><code>count(query)</code></td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td><code>count_ops([visual])</code></td>
<td>wrapper for count_ops that returns the operation count.</td>
</tr>
<tr>
<td><code>diff(*symbols, **kw_args)</code></td>
<td>Expand an expression using hints.</td>
</tr>
<tr>
<td><code>doit(**hints)</code></td>
<td></td>
</tr>
<tr>
<td><code>dummy_eq(other[, symbol])</code></td>
<td>Compare two expressions and handle dummy symbols.</td>
</tr>
<tr>
<td><code>equals(other[, failing_expression])</code></td>
<td>Return True if self == other, False if it doesn’t, or None. If</td>
</tr>
<tr>
<td><code>evalf([n, subs, maxn, chop, strict, quad, ...])</code></td>
<td>Evaluate the given formula to an accuracy of n digits.</td>
</tr>
<tr>
<td><code>expand(*args, **kw_args)</code></td>
<td>Expand an expression using hints.</td>
</tr>
<tr>
<td><code>extract_additively(c)</code></td>
<td>Return self - c if it’s possible to subtract c from self and</td>
</tr>
<tr>
<td><code>extract_branch_factor([allow_half])</code></td>
<td>Try to write self as exp_polar(2<em>pi</em>I*n)*z in a nice way.</td>
</tr>
<tr>
<td><code>extract_multiplicatively(c)</code></td>
<td>Return None if it’s not possible to make self in the form</td>
</tr>
<tr>
<td><code>factor(*gens, **args)</code></td>
<td>See the factor() function in sympy.polys.polytools</td>
</tr>
<tr>
<td><code>find(query[, group])</code></td>
<td>Find all subexpressions matching a query.</td>
</tr>
<tr>
<td><code>fromiter(args, **assumptions)</code></td>
<td>Create a new object from an iterable.</td>
</tr>
<tr>
<td><code>getO()</code></td>
<td>Returns the additive O(..) symbol if there is one, else None.</td>
</tr>
<tr>
<td><code>getn()</code></td>
<td>Returns the order of the expression.</td>
</tr>
<tr>
<td><code>has(*args, **kw_args)</code></td>
<td>Test whether any subexpression matches any of the patterns.</td>
</tr>
<tr>
<td><code>integrate(*args, **kwargs)</code></td>
<td>See the integrate function in sympy.integrals</td>
</tr>
<tr>
<td><code>invert(g)</code></td>
<td>See the invert function in sympy.polys</td>
</tr>
<tr>
<td><code>is_constant(*wrt, **flags)</code></td>
<td>Return True if self is a polynomial in syms and False otherwise.</td>
</tr>
<tr>
<td><code>is_hypergeometric(k)</code></td>
<td></td>
</tr>
<tr>
<td><code>is_rational_function(*syms)</code></td>
<td>Test whether function is a ratio of two polynomials in the given symbols, syms.</td>
</tr>
<tr>
<td><code>iter_basic_args()</code></td>
<td>Iterates arguments of self.</td>
</tr>
<tr>
<td><code>leadterm(x)</code></td>
<td>Returns the leading term a*x**b as a tuple (a, b).</td>
</tr>
<tr>
<td><code>limit(x, xlim[, dir])</code></td>
<td>Compute limit x-&gt;xlim.</td>
</tr>
<tr>
<td><code>lseries([x, x0, dir])</code></td>
<td>Wrapper for series yielding an iterator of the terms of the series.</td>
</tr>
<tr>
<td><code>match(pattern)</code></td>
<td>Pattern matching.</td>
</tr>
<tr>
<td><code>matches(expr, repl_dict)</code></td>
<td>Evaluate the given formula to an accuracy of n digits.</td>
</tr>
<tr>
<td><code>nsimplify([constants, tolerance, full])</code></td>
<td>See the nsimplify function in sympy.simplify</td>
</tr>
<tr>
<td><code>powseries([deep, combine])</code></td>
<td>See the powseries function in sympy.simplify</td>
</tr>
<tr>
<td><code>primitive()</code></td>
<td>Return the positive Rational that can be extracted non-recursively</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>radsimp()</td>
<td>See the radsimp function in sympy.simplify</td>
</tr>
<tr>
<td>ratsimp()</td>
<td>See the ratsimp function in sympy.simplify</td>
</tr>
<tr>
<td>refine(assumption)</td>
<td>See the refine function in sympy.assumptions</td>
</tr>
<tr>
<td>removeO()</td>
<td>Removes the additive O(...) symbol if there is one</td>
</tr>
<tr>
<td>replace(query, value[, map])</td>
<td>Replace matching subexpressions of self with value.</td>
</tr>
<tr>
<td>rewrite(*args, **hints)</td>
<td>Rewrites expression containing applications of functions of one kind in terms of</td>
</tr>
<tr>
<td>round([p])</td>
<td>Return x rounded to the given decimal place.</td>
</tr>
<tr>
<td>separate([deep, force])</td>
<td>See the separate function in sympy.simplify</td>
</tr>
<tr>
<td>series([x, x0, n, dir])</td>
<td>Series expansion of “self” around x = x0 yielding either terms of</td>
</tr>
<tr>
<td>simplify()</td>
<td>See the simplify function in sympy.simplify</td>
</tr>
<tr>
<td>sort_key(*args, **kw_args)</td>
<td></td>
</tr>
<tr>
<td>subs(*args, **kwargs)</td>
<td>Substitutes old for new in an expression after sympifying args.</td>
</tr>
<tr>
<td>together(*args, **kwargs)</td>
<td>See the together function in sympy.polys</td>
</tr>
<tr>
<td>transpose()</td>
<td></td>
</tr>
<tr>
<td>trigsimp([deep, recursive])</td>
<td>See the trigsimp function in sympy.simplify</td>
</tr>
<tr>
<td>xreplace()</td>
<td></td>
</tr>
</tbody>
</table>

---

### Notes

Never use self._args, always use self.args. Only use _args in __new__ when creating a new function. Don’t override .args() from Basic (so that it’s easy to change the interface in the future if needed).

### Examples

```python
>>> from sympy import symbols, cot
>>> from sympy.abc import x, y

>>> cot(x).args
(x,)

>>> cot(x).args[0]
x

>>> (x+y).args
(x, y)

>>> (x+y).args[1]
y
```

**args_cnc** (cset=False, warn=True)

Return [commutative factors, non-commutative factors] of self.
self is treated as a Mul and the ordering of the factors is maintained. If \texttt{cset} is True the commutative factors will be returned in a set. If there were repeated factors (as may happen with an unevaluated Mul) then an error will be raised unless it is explicitly suppressed by setting \texttt{warn} to False.

Note: -1 is always separated from a Number.

```python
>>> from sympy import symbols, oo
>>> A, B = symbols(’A B’, commutative=0)
>>> x, y = symbols(’x y’)
>>> (-2*x*y).args_cnc()
[[-1, 2, x, y], []]
>>> (-2.5*x).args_cnc()
[[-1, 2.5, x], []]
>>> (-2*x*A*B*y).args_cnc()
[[-1, 2, x, y], [A, B]]
>>> (-2*x*y).args_cnc(cset=True)
[set([[-1, 2, x, y]], []]
```

The arg is always treated as a Mul:

```python
>>> (-2 + x + A).args_cnc()
[[], [x - 2 + A]]
>>> (-oo).args_cnc()  # -oo is a singleton
[[-1, oo], []]
```

\texttt{as\_base\_exp()}

\texttt{as\_coeff\_Add()}

Efficiently extract the coefficient of a summation.

\texttt{as\_coeff\_Mul(rational=False)}

Efficiently extract the coefficient of a product.

\texttt{as\_coeff\_add(*deps)}

Return the tuple (c, args) where self is written as an Add, a.

\(c\) should be a Rational added to any terms of the Add that are independent of \texttt{deps}.

\texttt{args} should be a tuple of all other terms of \texttt{a}; \texttt{args} is empty if \texttt{self} is a Number or if \texttt{self} is independent of \texttt{deps} (when given).

This should be used when you don’t know if \texttt{self} is an Add or not but you want to treat \texttt{self} as an Add or if you want to process the individual arguments of the tail of \texttt{self} as an Add.

- if you know \texttt{self} is an Add and want only the head, use \texttt{self.args[0]};
- if you don’t want to process the arguments of the tail but need the tail then use \texttt{self.as\_two\_terms()} which gives the head and tail.
- if you want to split \texttt{self} into an independent and dependent parts use \texttt{self.as\_independent(*deps)}

```python
>>> from sympy import S
>>> from sympy.abc import x, y
>>> (S(3)).as_coeff_add()
(3, ())
>>> (3 + x).as_coeff_add()
(3, (x,))
>>> (3 + x + y).as_coeff_add(x)
(y + 3, (x,))
>>> (3 + y).as_coeff_add(x)
(y + 3, ())
```
as_coeff_exponent(x)
c*x**e -> c,e where x can be any symbolic expression.

as_coeff_factors(*deps)
This method is deprecated. Use .as_coeff_add() instead.

as_coeff_mul(*deps)
Return the tuple (c, args) where self is written as a Mul, m.

c should be a Rational multiplied by any terms of the Mul that are independent of deps.

args should be a tuple of all other terms of m; args is empty if self is a Number or if self is independent of
deps (when given).

This should be used when you don’t know if self is a Mul or not but you want to treat self as a Mul or if
you want to process the individual arguments of the tail of self as a Mul.

• if you know self is a Mul and want only the head, use self.args[0];

• if you don’t want to process the arguments of the tail but need the tail then use self.as_two_terms()
which gives the head and tail;

• if you want to split self into an independent and dependent parts use

self.as_independent(*deps)

>>> from sympy import S
>>> from sympy.abc import x, y
>>> (S(3)).as_coeff_mul()
(3, ())
>>> (3*x*y).as_coeff_mul()
(3, (x, y))
>>> (3*x*y).as_coeff_mul(x)
(3*y, (x,))
>>> (3*y).as_coeff_mul(x)
(3*y, ())

as_coeff_terms(*deps)
This method is deprecated. Use .as_coeff_mul() instead.

as_coefficient(expr)
Extracts symbolic coefficient at the given expression. In other words, this functions separates ‘self’ into
the product of ‘expr’ and ‘expr’-free coefficient. If such separation is not possible it will return None.

See Also:
coeff

Examples

>>> from sympy import E, pi, sin, I, symbols
>>> from sympy.abc import x, y

>>> E.as_coefficient(E)
1
>>> (2*E).as_coefficient(E)
2
>>> (2*sin(E)*E).as_coefficient(E)

>>> (2*E + x*E).as_coefficient(E)
x + 2
>>> (2*E*x + x).as_coefficient(E)
as_coefficient(E)

```python
>>> (E*(x + 1) + x).as_coefficient(E)
```

```python
>>> (2*pi*I).as_coefficient(pi*I)
2
>>> (2*I).as_coefficient(pi*I)
```

**as_coefficients_dict()**

Return a dictionary mapping terms to their Rational coefficient. Since the dictionary is a defaultdict, inquiries about terms which were not present will return a coefficient of 0. If an expression is not an Add it is considered to have a single term.

**Examples**

```python
>>> from sympy.abc import a, x
>>> (3*x + a*x + 4).as_coefficients_dict()
{1: 4, x: 3, a*x: 1}
>>> _[a]
0
>>> (3*a*x).as_coefficients_dict()
{a*x: 3}
```

**as_content_primitive**(radical=False)

This method should recursively remove a Rational from all arguments and return that (content) and the new self (primitive). The content should always be positive and `Mul(*foo.as_content_primitive()) == foo`. The primitive need not be in canonical form and should try to preserve the underlying structure if possible (i.e. expand_mul should not be applied to self).

**Examples**

```python
>>> from sympy import sqrt
>>> from sympy.abc import x, y, z

>>> eq = 2 + 2*x + 2*y*(3 + 3*y)

The as_content_primitive function is recursive and retains structure:

```python
>>> eq.as_content_primitive()
(2, x + 3*y*(y + 1) + 1)
```

Integer powers will have Rationals extracted from the base:

```python
>>> ((2 + 6*x)**2).as_content_primitive()
(4, (3*x + 1)**2)
>>> ((2 + 6*x)**(2*y)).as_content_primitive()
(1, (2*(3*x + 1))**(2*y))
```

Terms may end up joining once their as_content_primitives are added:

```python
>>> ((5*(x*(1 + y)) + 2*x*(3 + 3*y))).as_content_primitive()
(11, x*(y + 1))
>>> ((3*(x*(1 + y)) + 2*x*(3 + 3*y))).as_content_primitive()
(9, x*(y + 1))
>>> ((3*(z*(1 + y)) + 2.0*x*(3 + 3*y))).as_content_primitive()
```
(1, 6.0*x*(y + 1) + 3*z*(y + 1))
>>> ((5*(x*(1 + y)) + 2.0*x*(3 + 3*y))**2).as_content_primitive()
(121, x**2*(y + 1)**2)
>>> ((5*(x*(1 + y)) + 2.0*x*(3 + 3*y))**2).as_content_primitive()
(1, 121.0*x**2*(y + 1)**2)

Radical content can also be factored out of the primitive:
>>> (2*sqrt(2) + 4*sqrt(10)).as_content_primitive(radical=True)
(2, sqrt(2)*(1 + 2*sqrt(5)))

as_dummy()

as_expr(*gens)

Convert a polynomial to a SymPy expression.

Examples

>>> from sympy import sin
>>> from sympy.abc import x, y

>>> f = (x**2 + x*y).as_poly(x, y)
>>> f.as_expr()
x**2 + x*y
>>> sin(x).as_expr()
sin(x)

as_independent(*deps, **hint)

A mostly naive separation of a Mul or Add into arguments that are not are dependent on deps. To obtain as complete a separation of variables as possible, use a separation method first, e.g.:

*separatevars() to change Mul, Add and Pow (including exp) into Mul
*expand(mul=True) to change Add or Mul into Add
*expand(log=True) to change log expr into an Add

The only non-naive thing that is done here is to respect noncommutative ordering of variables.

The returned tuple (i, d) has the following interpretation:

*i will have no variable that appears in deps
*d will be 0 or else have terms that contain variables that are in deps
*if self is an Add then self = i + d
*if self is a Mul then self = i*d
*if self is anything else, either tuple (self, S.One) or (S.One, self) is returned.

To force the expression to be treated as an Add, use the hint as_Add=True

Examples

– self is an Add

>>> from sympy import sin, cos, exp
>>> from sympy.abc import x, y, z
>>> (x + x*y).as_independent(x)
(0, x*y + x)
>>> (x + x*y).as_independent(y)
(x, x*y)
>>> (2*x*sin(x) + y + x + z).as_independent(x)
(y + z, 2*x*sin(x) + x)
>>> (2*x*sin(x) + y + x + z).as_independent(x, y)
(z, 2*x*sin(x) + x + y)

– self is a Mul

>>> (x*sin(x)*cos(y)).as_independent(x)
(cos(y), x*sin(x))

non-commutative terms cannot always be separated out when self is a Mul

>>> from sympy import symbols
>>> n1, n2, n3 = symbols('n1 n2 n3', commutative=False)

>>> (n1 + n1*n2).as_independent(n2)
(n1, n1*n2)

>>> (n2*n1 + n1*n2).as_independent(n2)
(0, n1*n2 + n2*n1)

>>> (n1*n2+n3).as_independent(n1)
(1, n1*n2*n3)

>>> (n1*n2+n3).as_independent(n2)
(1, n2*n3)

>>> ((x-n1)*(x-y)).as_independent(x)
(1, (x - y)*(x - n1))

– self is anything else:

>>> (sin(x)).as_independent(x)
(1, sin(x))

>>> (sin(x)).as_independent(y)
(sin(x), 1)

>>> exp(x+y).as_independent(x)
(1, exp(x + y))

– force self to be treated as an Add:

>>> (3*x).as_independent(x, as_Add=True)
(0, 3*x)

– force self to be treated as a Mul:

>>> (3+x).as_independent(x, as_Add=False)
(1, x + 3)

>>> (-3+x).as_independent(x, as_Add=False)
(1, x - 3)

Note how the below differs from the above in making the constant on the dep term positive.

>>> (y*(-3+x)).as_independent(x)
(y, x - 3)

– use .as_independent() for true independence testing instead of .has(). The former considers only symbols in the free symbols while the latter considers all symbols.
```python
>>> from sympy import Integral
>>> I = Integral(x, (x, 1, 2))
>>> I.has(x)
True
>>> x in I.free_symbols
False
>>> I.as_independent(x) == (I, 1)
True
>>> (I + x).as_independent(x) == (I, x)
True
```

Note: when trying to get independent terms, a separation method might need to be used first. In this case, it is important to keep track of what you send to this routine so you know how to interpret the returned values.

```python
>>> from sympy import separatevars, log
>>> separatevars(exp(x+y)).as_independent(x)
(exp(y), exp(x))
>>> (x + x*y).as_independent(y)
(x, x*y)
```

See also: `separatevars()`, `expand(log=True)`, `as_two_terms()`, `as_coeff_add()`, `as_coeff_mul()`

**as_leading_term** (*args, **kw_args*)

Returns the leading (nonzero) term of the series expansion of self.

The _eval_as_leading_term routines are used to do this, and they must always return a non-zero value.

**Examples**

```python
>>> from sympy.abc import x
>>> (1 + x + x**2).as_leading_term(x)
1
>>> (1/x**2 + x + x**2).as_leading_term(x)
x**(-2)
```

**as_numer_denom()**

expression -> a/b -> a, b

This is just a stub that should be defined by an object’s class methods to get anything else.

See Also:

- `normal` return a/b instead of a, b

**as_ordered_factors** (*order=None*)

Return list of ordered factors (if Mul) else [self].
as_ordered_terms(order=None, data=False)
Transform an expression to an ordered list of terms.

Examples

```python
>>> from sympy import sin, cos
>>> from sympy.abc import x, y

>>> (sin(x)**2*cos(x) + sin(x)**2 + 1).as_ordered_terms()
[sin(x)**2*cos(x), sin(x)**2, 1]
```

as_poly(*gens, **args)
Converts self to a polynomial or returns None.

```python
>>> from sympy import Poly, sin
>>> from sympy.abc import x, y

>>> print (x**2 + x*y).as_poly()
Poly(x**2 + x*y, x, y, domain='ZZ')

>>> print (x**2 + x*y).as_poly(x, y)
Poly(x**2 + x*y, x, y, domain='ZZ')

>>> print (x**2 + sin(y)).as_poly(x, y)
None
```

as_powers_dict()
Return self as a dictionary of factors with each factor being treated as a power. The keys are the bases of the factors and the values, the corresponding exponents. The resulting dictionary should be used with caution if the expression is a Mul and contains non-commutative factors since the order that they appeared will be lost in the dictionary.

as_real_imag(deep=True, **hints)

as_terms()
Transform an expression to a list of terms.

assumptions0

atoms(*types)
Returns the atoms that form the current object.

By default, only objects that are truly atomic and can’t be divided into smaller pieces are returned: symbols, numbers, and number symbols like I and pi. It is possible to request atoms of any type, however, as demonstrated below.

Examples

```python
>>> from sympy import Number, NumberSymbol, Symbol

>>> (1 + x + 2*sin(y + I*pi)).atoms(Symbol)
set([x, y])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Number)
set([1, 2])
```
>>> (1 + x + 2*sin(y + I*pi)).atoms(Number, NumberSymbol)
set([1, 2, pi])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Number, NumberSymbol, I)
set([1, 2, I, pi])

Note that I (imaginary unit) and zoo (complex infinity) are special types of number symbols and are not part of the NumberSymbol class.

The type can be given implicitly, too:

>>> (1 + x + 2*sin(y + I*pi)).atoms(x) # x is a Symbol
set([x, y])

Be careful to check your assumptions when using the implicit option since S(1).is_Integer = True but type(S(1)) is One, a special type of sympy atom, while type(S(2)) is type Integer and will find all integers in an expression:

>>> from sympy import S
>>> (1 + x + 2*sin(y + I*pi)).atoms(S(1))
set([1])

Finally, arguments to atoms() can select more than atomic atoms: any sympy type (loaded in core/__init__.py) can be listed as an argument and those types of “atoms” as found in scanning the arguments of the expression recursively:

>>> from sympy import Function, Mul
>>> from sympy.core.function import AppliedUndef

>>> f = Function('f')

>>> (1 + f(x) + 2*sin(y + I*pi)).atoms(Function)
set([f(x), sin(y + I*pi)])

>>> (1 + f(x) + 2*sin(y + I*pi)).atoms(AppliedUndef)
set([f(x)])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Mul)
set([I*pi, 2*sin(y + I*pi)])

cancel(*gens, **args)
See the cancel function in sympy.polys

classmethod class_key()

coeff(x, n=1, right=False)
Returns the coefficient from the term containing “x**n” or None if there is no such term. If n is zero then all terms independent of x will be returned.

When x is noncommutative, the coeff to the left (default) or right of x can be returned. The keyword ‘right’ is ignored when x is commutative.

See Also:

as_coeff_Add a method to separate the additive constant from an expression

as_coeff_Mul a method to separate the multiplicative constant from an expression

as_independent a method to separate x dependent terms/factors from others
Examples

```python
>>> from sympy import symbols
>>> from sympy.abc import x, y, z

You can select terms that have an explicit negative in front of them:

```python
>>> (-x + 2*y).coeff(-1)
x
```  

```python
>>> (x - 2*y).coeff(-1)
2*y
```  

You can select terms with no Rational coefficient:

```python
>>> (x + 2*y).coeff(1)
x
```  

```python
>>> (3 + 2*x + 4*x**2).coeff(1)
0
```  

You can select terms independent of x by making n=0; in this case expr.as_independent(x)[0] is returned (and 0 will be returned instead of None):

```python
>>> (3 + 2*x + 4*x**2).coeff(x, 0)
3
```  

```python
>>> eq = ((x + 1)**3).expand() + 1
```  

```python
>>> eq
x**3 + 3*x**2 + 3*x + 2
```  

```python
>>> [eq.coeff(x, i) for i in reversed(range(4))]
[1, 3, 3, 2]
```  

```python
>>> eq -= 2
```  

```python
>>> [eq.coeff(x, i) for i in reversed(range(4))]
[1, 3, 3, 0]
```  

You can select terms that have a numerical term in front of them:

```python
>>> (-x - 2*y).coeff(2)
-y
```  

```python
>>> from sympy import sqrt
>>> (x + sqrt(2)*x).coeff(sqrt(2))
x
```  

The matching is exact:

```python
>>> (3 + 2*x + 4*x**2).coeff(x)
2
```  

```python
>>> (3 + 2*x + 4*x**2).coeff(x**2)
4
```  

```python
>>> (3 + 2*x + 4*x**2).coeff(x**3)
0
```  

```python
>>> (z*(x + y)**2).coeff((x + y)**2)
z
```  

```python
>>> (z*(x + y)**2).coeff(x + y)
0
```  

In addition, no factoring is done, so 1 + z*(1 + y) is not obtained from the following:

```python
>>> (x + z*(x + x*y)).coeff(x)
1
```  

If such factoring is desired, factor_terms can be used first:
>>> from sympy import factor_terms
>>> factor_terms(x + z*(x + x*y)).coeff(x)
z*(y + 1) + 1

>>> n, m, o = symbols(’n m o’, commutative=False)
>>> n.coeff(n)
1
>>> (3*n).coeff(n)
3
>>> (n*m + m*n*m).coeff(n) # = (1 + m)*n*m
1 + m
>>> (n*m + m*n*m).coeff(n, right=True) # = (1 + m)*n*m
m

If there is more than one possible coefficient 0 is returned:

>>> (n*m + m*n).coeff(n)
0

If there is only one possible coefficient, it is returned:

>>> (n*m + x*m*n).coeff(m*n)
x
>>> (n*m + x*m*n).coeff(m*n, right=True)
1

**collect**(syms, func=None, evaluate=True, exact=False, distribute_order_term=True)

See the collect function in sympy.simplify

**combsimp**()

See the combsimp function in sympy.simplify

**compare**(other)

Return -1, 0, 1 if the object is smaller, equal, or greater than other.

Not in the mathematical sense. If the object is of a different type from the “other” then their classes are ordered according to the sorted_classes list.

**Examples**

>>> from sympy.abc import x, y
>>> x.compare(y)
-1
>>> x.compare(x)
0
>>> y.compare(x)
1

**static compare_pretty**(args, **kwargs)

Is a > b in the sense of ordering in printing?

**THIS FUNCTION IS DEPRECATED. Use default_sort_key instead.**

yes ...... return 1
no ...... return -1
equal ... return 0

**Strategy:**
It uses Basic.compare as a fallback, but improves it in many cases, like \(x^3, x^4, O(x^3)\) etc. In those simple cases, it just parses the expression and returns the “sane” ordering such as:

\[1 < x < x^2 < x^3 < O(x^4)\] etc.

**Examples**

```python
>>> from sympy.abc import x
>>> from sympy import Basic, Number

>>> Basic._compare_pretty(x, x**2)
-1
>>> Basic._compare_pretty(x**2, x**2)
0
>>> Basic._compare_pretty(x**3, x**2)
1
>>> Basic._compare_pretty(Number(1, 2), Number(1, 3))
1
>>> Basic._compare_pretty(Number(0), Number(-1))
1
```

**compute_leading_term** \((x, \text{skip_abs}=False, \text{logx}=None)\)

`as_leading_term` is only allowed for results of `.series()` This is a wrapper to compute a series first. If `skip_abs` is true, the absolute term is assumed to be zero. (This is necessary because sometimes it cannot be simplified to zero without a lot of work, but is still known to be zero. See `log._eval_nseries` for an example.) If `skip_log` is true, `log(x)` is treated as an independent symbol. (This is needed for the gruntz algorithm.)

**conjugate**

**copy**

**could_extract_minus_sign**

Canonical way to choose an element in the set \{e, -e\} where e is any expression. If the canonical element is e, we have e.could_extract_minus_sign() == True, else e.could_extract_minus_sign() == False. For any expression, the set \{e.could_extract_minus_sign(), (-e).could_extract_minus_sign()\} must be \{True, False\}.

```python
>>> from sympy.abc import x, y

>>> (x-y).could_extract_minus_sign() != (y-x).could_extract_minus_sign()
True
```

**count** \((query)\)

Count the number of matching subexpressions.

**count_ops** \((\text{visual}=None)\)

Wrapper for count_ops that returns the operation count.

**default_assumptions** = {}  

**diff** \((\text{*symbols, **assumptions})\)

**doit** \((\text{**hints})\)

**dummy_eq** \((\text{other, symbol}=None)\)

Compare two expressions and handle dummy symbols.
Examples

```python
>>> from sympy import Dummy
>>> from sympy.abc import x, y

>>> u = Dummy('u')

>>> (u**2 + 1).dummy_eq(x**2 + 1)
True
>>> (u**2 + 1) == (x**2 + 1)
False

>>> (u**2 + y).dummy_eq(x**2 + y, x)
True
>>> (u**2 + y).dummy_eq(x**2 + y, y)
False
```

dummy_index
equals(other, failing_expression=False)
Return True if self == other, False if it doesn’t, or None. If failing_expression is True then the expression which did not simplify to a 0 will be returned instead of None.

If self is a Number (or complex number) that is not zero, then the result is False.

If self is a number and has not evaluated to zero, evafl will be used to test whether the expression evaluates to zero. If it does so and the result has significance (i.e. the precision is either -1, for a Rational result, or is greater than 1) then the evafl value will be used to return True or False.

evalf(n=15, subs=None, maxn=100, chop=False, strict=False, quad=None, verbose=False)
Evaluate the given formula to an accuracy of n digits. Optional keyword arguments:

- subs=<dict> Substitute numerical values for symbols, e.g. subs={x:3, y:1+pi}.
- maxn=<integer> Allow a maximum temporary working precision of maxn digits (default=100)
- chop=<bool> Replace tiny real or imaginary parts in subresults by exact zeros (default=False)
- strict=<bool> Raise PrecisionExhaused if any subresult fails to evaluate to full accuracy, given the available maxprec (default=False)
- quad=<str> Choose algorithm for numerical quadrature. By default, tanh-sinh quadrature is used. For oscillatory integrals on an infinite interval, try quad='osc'.
- verbose=<bool> Print debug information (default=False)

expand(*args, **kw_args)
Expand an expression using hints.

See the docstring of the expand() function in sympy.core.function for more information.

extract_additively(c)
Return self - c if it’s possible to subtract c from self and make all matching coefficients move towards zero, else return None.

See Also:
extract_multiplicatively, coeff, as_coefficient
Examples

```python
>>> from sympy import S
>>> from sympy.abc import x, y

>>> e = 2*x + 3
>>> e.extract_additively(x + 1)
x + 2
>>> e.extract_additively(3*x)
>>> e.extract_additively(4)
>>> (y*(x + 1)).extract_additively(x + 1)
>>> ((x + 1)*(x + 2*y + 1) + 3).extract_additively(x + 1)
```

Sometimes auto-expansion will return a less simplified result than desired; gcd_terms might be used in such cases:

```python
>>> from sympy import gcd_terms

>>> (4*x*(y + 1) + y).extract_additively(x)
4*x*(y + 1) + x*(4*y + 3) - x*(4*y + 4) + y
>>> gcd_terms(_)
x*(4*y + 3) + y
```

**extract_branch_factor** *(allow_half=False)*

Try to write self as \( \exp(2\pi i n) z \) in a nice way. Return \((z, n)\).

```python
>>> from sympy import exp_polar, I, pi
>>> from sympy.abc import x, y

>>> exp_polar(I*pi).extract_branch_factor()
(exp_polar(I*pi), 0)
```

If allow_half is True, also extract \( \exp(\pi i/2) \):

```python
>>> exp_polar(I*pi).extract_branch_factor(allow_half=True)
(1, 1/2)
```

**extract_multiplicatively** *(c)*

Return None if it’s not possible to make self in the form \( c * \text{something} \) in a nice way, i.e. preserving the properties of arguments of self.

```python
>>> from sympy import symbols, Rational
```
>>> x, y = symbols('x, y', real=True)

>>> ((x*y)**3).extract_multiplicatively(x**2 * y)
x*y**2

>>> ((x*y)**3).extract_multiplicatively(x**4 * y)
2*x

>>> (2*x).extract_multiplicatively(3)
3

>>> (Rational(1,2)*x).extract_multiplicatively(3)
x/6

factor(*gens, **args)
See the factor() function in sympy.polys.polytools

find(query, group=False)
Find all subexpressions matching a query.

free_symbols

classmethod fromiter(args, **assumptions)
Create a new object from an iterable.
This is a convenience function that allows one to create objects from any iterable, without having to convert to a list or tuple first.

Examples

>>> from sympy import Tuple
>>> Tuple.fromiter(i for i in xrange(5))
(0, 1, 2, 3, 4)

func
The top-level function in an expression.
The following should hold for all objects:

>>> x == x.func(*x.args)

Examples

>>> from sympy.abc import x
>>> a = 2*x
>>> a.func
<class 'sympy.core.mul.Mul'>
>>> a.args
(2, x)
>>> a.func(*a.args)
2*x
>>> a == a.func(*a.args)
True

getO()
Returns the additive O(.) symbol if there is one, else None.
getn()
Returns the order of the expression.
The order is determined either from the O(...) term. If there is no O(...) term, it returns None.

Examples

```python
>>> from sympy import O
>>> from sympy.abc import x

>>> (1 + x + O(x**2)).getn()
2
>>> (1 + x).getn()
```

has(*args, **kw_args)
Test whether any subexpression matches any of the patterns.

Examples

```python
>>> from sympy import sin, S
>>> from sympy.abc import x, y, z

>>> (x**2 + sin(x*y)).has(z)
False
>>> (x**2 + sin(x*y)).has(x, y, z)
True
>>> x.has(x)
True

Note that expr.has(*patterns) is exactly equivalent to any(expr.has(p) for p in patterns). In particular, False is returned when the list of patterns is empty.

>>> x.has()
False
```

integrate(*args, **kwargs)
See the integrate function in sympy.integrals

invert(g)
See the invert function in sympy.polys

is_Add = False
is_AlgebraicNumber = False
is_Atom = True
is_Boolean = False
is_Derivative = False
is_Dummy = True
is_Equality = False
is_Float = False
is_Function = False
is_Integer = False
is_Matrix = False
is_Mul = False
is_Not = False
is_Number = False
is_NumberSymbol = False
is_Order = False
is_Piecewise = False
is_Poly = False
is_Pow = False
is_Rational = False
is_Real
    Deprecated alias for is_Float
is_Relational = False
is_Symbol = True
is_Wild = False
is_antihermitian
is_bounded
is_commutative
is_comparable = False
is_complex
is_composite
is_constant (*wrt. **flags)
is_even
is_finite
is_hermitian
is_hypergeometric(k)
is_imaginary
is_infinities
is_integer
is_irrational
is_negative
is_noninteger
is_nonnegative
is_nonpositive
is_nonzero
is_number
is_odd
is_polar

is_polynomial(*syms)

Return True if self is a polynomial in syms and False otherwise.

This checks if self is an exact polynomial in syms. This function returns False for expressions that are “polynomials” with symbolic exponents. Thus, you should be able to apply polynomial algorithms to expressions for which this returns True, and Poly(expr, *syms) should work only if and only if expr.is_polynomial(*syms) returns True. The polynomial does not have to be in expanded form. If no symbols are given, all free symbols in the expression will be used.

This is not part of the assumptions system. You cannot do Symbol('z', polynomial=True).

Examples

```python
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> ((x**2 + 1)**4).is_polynomial(x)
True
>>> ((x**2 + 1)**4).is_polynomial()
True
>>> (2*x + 1).is_polynomial(x)
False

>>> n = Symbol('n', nonnegative=True, integer=True)
>>> (x**n + 1).is_polynomial(x)
False
```

This function does not attempt any nontrivial simplifications that may result in an expression that does not appear to be a polynomial to become one.

```python
>>> from sympy import sqrt, factor, cancel
>>> y = Symbol('y', positive=True)
>>> a = sqrt(y**2 + 2*y + 1)
>>> a.is_polynomial(y)
False
>>> factor(a)
y + 1
>>> factor(a).is_polynomial(y)
True

>>> b = (y**2 + 2*y + 1)/(y + 1)
>>> b.is_polynomial(y)
False
>>> cancel(b)
y + 1
>>> cancel(b).is_polynomial(y)
True
```

See also .is_rational_function()

is_positive

is_prime

is_rational

is_rational_function(*syms)

Test whether function is a ratio of two polynomials in the given symbols, syms. When syms is not given,
all free symbols will be used. The rational function does not have to be in expanded or in any kind of canonical form.

This function returns False for expressions that are “rational functions” with symbolic exponents. Thus, you should be able to call .as_numer_denom() and apply polynomial algorithms to the result for expressions for which this returns True.

This is not part of the assumptions system. You cannot do Symbol(‘z’, rational_function=True).

Examples

```python
>>> from sympy import Symbol, sin
>>> from sympy.abc import x, y

>>> (x/y).is_rational_function()
True

>>> (x**2).is_rational_function()
True

>>> (x/sin(y)).is_rational_function(y)
False

>>> n = Symbol(‘n’, integer=True)

>>> (x**n + 1).is_rational_function(x)
False
```

This function does not attempt any nontrivial simplifications that may result in an expression that does not appear to be a rational function to become one.

```python
>>> from sympy import sqrt, factor, cancel

>>> y = Symbol(‘y’, positive=True)

>>> a = sqrt(y**2 + 2*y + 1)/y

>>> a.is_rational_function(y)
False

>>> factor(a)
(y + 1)/y

>>> factor(a).is_rational_function(y)
True
```

See also is_rational_function().

is_real

is_unbounded

is_zero

iter_basic_args()

Iterates arguments of self.

Examples

```python
>>> from sympy.abc import x

>>> a = 2*x

>>> a.iter_basic_args()
<...iterator object at 0x...>
```
leadterm (x)
Returns the leading term a*x**b as a tuple (a, b).

>>> from sympy.abc import x
>>> (1+x+x**2).leadterm(x)
(1, 0)
>>> (1/x**2+x+x**2).leadterm(x)
(1, -2)

limit (x, xlim, dir='+')</br>Compute limit x->xlim.

lseries (x=None, x0=0, dir='+')</br>Wrapper for series yielding an iterator of the terms of the series.</br>Note: an infinite series will yield an infinite iterator. The following, for example, will never terminate. It will just keep printing terms of the sin(x) series:

```python
for term in sin(x).lseries(x):
    print term
```

The advantage of lseries() over nseries() is that many times you are just interested in the next term in the series (i.e. the first term for example), but you don’t know how many you should ask for in nseries() using the “n” parameter.

See also nseries().

match (pattern)
Pattern matching.
Wild symbols match all.
Return None when expression (self) does not match with pattern. Otherwise return a dictionary such that:

```
expression.xreplace(expression.match(pattern)) == expression
```

Examples

```python
>>> from sympy import symbols, Wild
>>> from sympy.abc import x, y
>>> p = Wild("p")
>>> q = Wild("q")
>>> r = Wild("r")
>>> e = (x+y)**(x+y)
>>> e.match(p**p)
{p_: x + y}
>>> e.match(p**q)
{p_: x + y, q_: x + y}
>>> e = (2*x)**2
>>> e.match(p*q*r)
{p_: 4, q_: x, r_: 2}
```
```python
>>> (p*q**r).xreplace(e.match(p*q**r))
4*x**2
```

matches(expr, repl_dict={})

n (n=15, subs=None, maxn=100, chop=False, strict=False, quad=None, verbose=False)

Evaluate the given formula to an accuracy of n digits. Optional keyword arguments:

- subs=<dict> Substitute numerical values for symbols, e.g. subs={x:3, y:1+pi}.
- maxn=<integer> Allow a maximum temporary working precision of maxn digits (default=100)
- chop=<bool> Replace tiny real or imaginary parts in subresults by exact zeros (default=False)
- strict=<bool> Raise PrecisionExhausted if any subresult fails to evaluate to full accuracy, given the available maxprec (default=False)
- quad=<str> Choose algorithm for numerical quadrature. By default, tanh-sinh quadrature is used. For oscillatory integrals on an infinite interval, try quad='osc'.
- verbose=<bool> Print debug information (default=False)

name

normal()

nseries (x=None, x0=0, n=6, dir='+', logx=None)

Wrapper to _eval_nseries if assumptions allow, else to series.

If x is given, x0 is 0, dir='+', and self has x, then _eval_nseries is called. This calculates “n” terms in the innermost expressions and then builds up the final series just by “cross-multiplying” everything out.

Advantage – it’s fast, because we don’t have to determine how many terms we need to calculate in advance.

Disadvantage – you may end up with less terms than you may have expected, but the O(x**n) term appended will always be correct and so the result, though perhaps shorter, will also be correct.

If any of those assumptions is not met, this is treated like a wrapper to series which will try harder to return the correct number of terms.

See also lseries().

nsimplify (constants=[], tolerance=None, full=False)

See the nsimplify function in sympy.simplify

powsimp (deep=False, combine='all')

See the powsimp function in sympy.simplify

primitive()

Return the positive Rational that can be extracted non-recursively from every term of self (i.e., self is treated like an Add). This is like the as_coeff_Mul() method but primitive always extracts a positive Rational (never a negative or a Float).

Examples

```python
>>> from sympy.abc import x
>>> (3*(x + 1)**2).primitive()
(3, (x + 1)**2)
>>> a = (6*x + 2); a.primitive()
(2, 3*x + 1)
>>> b = (x/2 + 3); b.primitive()
(1/2, x + 6)
```
>>> (a*b).primitive() == (1, a*b)
True

**radsimp()**
See the radsimp function in sympy.simplify

**ratsimp()**
See the ratsimp function in sympy.simplify

**refine(assumption=True)**
See the refine function in sympy.assumptions

**removeO()**
Removes the additive $O(.)$ symbol if there is one

**replace(query, value, map=False)**
Replace matching subexpressions of `self` with `value`.

If `map = True` then also return the mapping `{old: new}` where `old` was a sub-expression found with query and `new` is the replacement value for it.

Traverses an expression tree and performs replacement of matching subexpressions from the bottom to the top of the tree. The list of possible combinations of queries and replacement values is listed below:

**See Also:**

**subs** substitution of subexpressions as defined by the objects themselves.

**xreplace** exact node replacement in expr tree; also capable of using matching rules

**Examples**

Initial setup

```python
>>> from sympy import log, sin, cos, tan, Wild
>>> from sympy.abc import x, y
```

```python
f = log(sin(x)) + tan(sin(x**2))
```

1. **type -> type** `obj.replace(sin, tan)`

```python
>>> f.replace(sin, cos)
log(cos(x)) + tan(cos(x**2))
>>> sin(x).replace(sin, cos, map=True)
(cos(x), {sin(x): cos(x)})
```

2. **type -> func** `obj.replace(sin, lambda arg: ...)`

```python
>>> f.replace(sin, lambda arg: sin(2*arg))
log(sin(2*x)) + tan(sin(2*x**2))
```

2.1. **expr -> expr** `obj.replace(sin(a), tan(a))`

```python
>>> a = Wild(‘a’)  # to match sin(a)
>>> f.replace(sin(a), tan(a))
log(tan(x)) + tan(tan(x**2))
```

2.2. **expr -> func** `obj.replace(sin(a), lambda a: ...)`

```python
>>> f.replace(sin(a), cos(a))
log(cos(x)) + tan(cos(x**2))
>>> f.replace(sin(a), lambda a: sin(2*a))
log(sin(2*x)) + tan(sin(2*x**2))
```

3.1. **func -> func**  
`obj.replace(lambda expr: ..., lambda expr: ...)`

```python
>>> g = 2*sin(x**3)
>>> g.replace(lambda expr: expr.is_Number, lambda expr: expr**2)
4*sin(x**9)
```

### rewrite(*args, **hints)**

Rewrites expression containing applications of functions of one kind in terms of functions of different kind. For example you can rewrite trigonometric functions as complex exponentials or combinatorial functions as gamma function.

As a pattern this function accepts a list of functions to to rewrite (instances of DefinedFunction class). As rule you can use string or a destination function instance (in this case rewrite() will use the str() function).

There is also possibility to pass hints on how to rewrite the given expressions. For now there is only one such hint defined called `deep`. When `deep` is set to False it will forbid functions to rewrite their contents.

```python
>>> from sympy import sin, exp, I
>>> from sympy.abc import x, y

>>> sin(x).rewrite(sin, exp)
-I*(exp(I*x) - exp(-I*x))/2
```

### round(p=0)

Return x rounded to the given decimal place.

If a complex number would results, apply round to the real and imaginary components of the number.

**Notes**

Do not confuse the Python built-in function, round, with the SymPy method of the same name. The former always returns a float (or raises an error if applied to a complex value) while the latter returns either a Number or a complex number:

```python
>>> isinstance(round(S(123), -2), Number)
False
>>> isinstance(S(123).round(-2), Number)
True
>>> isinstance((3*I).round(), Mul)
True
>>> isinstance((1 + 3*I).round(), Add)
True
```

**Examples**

```python
>>> from sympy import pi, E, I, S, Add, Mul, Number

>>> S(10.5).round()
11.
>>> pi.round()
3.
```
The round method has a chopping effect:

```python
>>> (2*pi + E*I).round()
6. + 3.*I
```
>>> cos(x).series(n=4)
1 - x**2/2 + O(x**4)

>>> e = cos(x + exp(y))
>>> e.series(y, n=2)
\cos(x + 1) - y\sin(x + 1) + O(y**2)

If n=None then an iterator of the series terms will be returned.

>>> term=cos(x).series(n=None)
>>> [term.next() for i in range(2)]
[1, -x**2/2]

For dir=+ (default) the series is calculated from the right and for dir=- the series from the left. For smooth functions this flag will not alter the results.

>>> abs(x).series(dir="+")
x

simplify()
See the simplify function in sympy.simplify

sort_key(*args, **kw_args)

subs(*args, **kwargs)
Substitutes old for new in an expression after sympifying args.

args is either:

- two arguments, e.g. foo.subs(old, new)
- one iterable argument, e.g. foo.subs(\{old, new\}). The iterable may be
  - an iterable container with \{(old, new)\} pairs. In this case the replacements are processed
    in the order given with successive patterns possibly affecting replacements already made.
  - a dict or set whose key/value items correspond to \{old/new\} pairs. In this case the old/new
    pairs will be sorted by op count and in case of a tie, by number of args and the default_sort_key. The resulting sorted list is then processed as an iterable container (see previous).

If the keyword simultaneous is True, the subexpressions will not be evaluated until all the substitutions have been made.

See Also:

replace replacement capable of doing wildcard-like matching, parsing of match, and conditional replacements
xreplace exact node replacement in expr tree; also capable of using matching rules

Examples
>>> from sympy import pi, exp
>>> from sympy.abc import x, y

>>> (1 + x*y).subs(x, pi)
pi*y + 1

>>> (1 + x*y).subs({x: pi, y: 2})
1 + 2*pi

>>> reps = [(y, x**2), (x, 2)]

>>> (x + y).subs(reps)
x**2 + 2

>>> (x**2 + x**4).subs(x**2, y)
y**2 + y

To replace only the x**2 but not the x**4, use xreplace:

>>> (x**2 + x**4).xreplace({x**2: y})
x**4 + y

To delay evaluation until all substitutions have been made, set the keyword simultaneous to True:

>>> (x/y).subs([{x: 0}, {y: 0}])
0

>>> (x/y).subs([{x: 0}, {y: 0}], simultaneous=True)
nan

This has the added feature of not allowing subsequent substitutions to affect those already made:

>>> ((x + y)/y).subs({x + y: y, y: x + y})
y/(x + y)

In order to obtain a canonical result, unordered iterables are sorted by count_op length, number of arguments and by the default_sort_key to break any ties. All other iterables are left unsorted.

>>> from sympy import sqrt, sin, cos, exp
>>> from sympy.abc import a, b, c, d, e

>>> A = (sqrt(sin(2*x)), a)
>>> B = (sin(2*x), b)
>>> C = (cos(2*x), c)
>>> D = (x, d)
>>> E = (exp(x), e)

>>> expr = sqrt(sin(2*x)) * sin(exp(x) * x) * cos(2*x) + sin(2*x)

>>> expr.subs(dict([A, B, C, D, E]))
a*c*sin(d*e) + b

together(*args, **kwargs)
See the together function in sympy.polys

transpose()

trigsimp(deep=False, recursive=False)
See the trigsimp function in sympy.simplify
52.2.2 Factor

class nipy.algorithms.statistics.formula.formulae.Factor (name, levels, char='b')

A qualitative variable in a regression model

A Factor is similar to R’s factor. The levels of the Factor can be either strings or ints.

Methods

__init__ (name, levels, char='b')

Parameters

name : str

levels : [str or int]

A sequence of strings or ints.

char : str

coefs

Coefficients in the linear regression formula.

design (input[, param, return_float, contrasts])

Construct the design matrix, and optional contrast matrices.

Parameters

input : np.recarray

Recarray including fields needed to compute the Terms in getparams(self.design_expr).

param : None or np.recarray

Recarray including fields that are not Terms in getparams(self.design_expr)

return_float : bool, optional

If True, return a np.float array rather than a np.recarray

contrasts : None or dict, optional

Contrasts. The items in this dictionary should be (str, Formula) pairs where a contrast matrix is constructed for each Formula by evaluating its design at the same parameters as self.design. If not None, then the return_float is set to True.

Returns

des : 2D array

design matrix

cmatrices : dict, optional
Dictionary with keys from `contrasts` input, and contrast matrices corresponding to `des` design matrix. Returned only if `contrasts` input is not None

**design_expr**

**dtype**
The dtype of the design matrix of the Formula.

**static fromcol**(col, name)
Create a Factor from a column array.

**Parameters**
- **col**: ndarray
  - an array with ndim==1
- **name**: str
  - name of the Factor

**Returns**
- **factor**: Factor

**Examples**

```python
>>> data = np.array([(3,'a'),(4,'a'),(5,'b'),(3,'b')], np.dtype([('x', np.float), ('y', 'S1')])
>>> f1 = Factor.fromcol(data['y'], 'y')
>>> f2 = Factor.fromcol(data['x'], 'x')
>>> d = f1.design(data)
>>> print d.dtype.descr
[('y_a', '<f8'), ('y_b', '<f8')]
>>> d = f2.design(data)
>>> print d.dtype.descr
[('x_3', '<f8'), ('x_4', '<f8'), ('x_5', '<f8')]
```

**static fromrec**(rec, keep=[], drop=[])
Construct Formula from recarray

For fields with a string-dtype, it is assumed that these are qualitative regressors, i.e. Factors.

**Parameters**
- **rec**: recarray
  - Recarray whose field names will be used to create a formula.
- **keep**: []
  - Field names to explicitly keep, dropping all others.
- **drop**: []
  - Field names to drop.

**get_term**(level)
Retrieve a term of the Factor...

**main_effect**

**mean**
Expression for the mean, expressed as a linear combination of terms, each with dummy variables in front.

**params**
The parameters in the Formula.

**stratify**(variable)
Create a new variable, stratified by the levels of a Factor.
**Parameters** variable : str or a simple sympy expression whose string representation are all lower or upper case letters, i.e. it can be interpreted as a name

**Returns** formula : Formula

Formula whose mean has one parameter named variable%d, for each level in self.levels

**Examples**

```python
>>> f = Factor('a', ['x', 'y'])
>>> sf = f.stratify('theta')
>>> sf.mean
_theta0*a_x + _theta1*a_y
```

**subs (old, new)**
Perform a sympy substitution on all terms in the Formula

Returns a new instance of the same class

**Parameters** old : sympy.Basic
The expression to be changed

new : sympy.Basic
The value to change it to.

**Returns** newf : Formula

**Examples**

```python
>>> s, t = [Term(l) for l in 'st']
>>> f, g = [sympy.Function(l) for l in 'fg']
>>> form = Formula([f(t), g(s)])
>>> newform = form.subs(g, sympy.Function('h'))
>>> newform.terms
array([f(t), h(s)], dtype=object)
>>> form.terms
array([f(t), g(s)], dtype=object)
```

**terms**
Terms in the linear regression formula.

### 52.2.3 FactorTerm

```python
class FactorTerm(nipy.algorithms.statistics.formula.formulae.FactorTerm
Bases: nipy.algorithms.statistics.formula.formulae.Term
```

Boolean Term derived from a Factor.

Its properties are the same as a Term except that its product with itself is itself.

**Methods**

```python
__call__(*args)
```
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>adjoint()</code></td>
<td>See the <code>adjoint</code> function in sympy.polys</td>
</tr>
<tr>
<td><code>apart(x)</code></td>
<td>Return <code>[commutative factors, non-commutative factors]</code> of self.</td>
</tr>
<tr>
<td><code>as_base_exp()</code></td>
<td>Efficiently extract the coefficient of a summation.</td>
</tr>
<tr>
<td><code>as_coeff_Add()</code></td>
<td>Efficiently extract the coefficient of a product.</td>
</tr>
<tr>
<td><code>as_coeff_Mul()</code></td>
<td>Return the tuple <code>(c, args)</code> where self is written as an Add, <code>a</code>.</td>
</tr>
<tr>
<td><code>as_coeff_add(*deps)</code></td>
<td>This method is deprecated.</td>
</tr>
<tr>
<td><code>as_coeff_exponent(x)</code></td>
<td><code>c*x**e -&gt; c, e</code> where <code>x</code> can be any symbolic expression.</td>
</tr>
<tr>
<td><code>as_coeff_factors(*deps)</code></td>
<td>Return a dictionary mapping terms to their Rational coefficient.</td>
</tr>
<tr>
<td><code>as_coeff_mul(*deps)</code></td>
<td>Return the tuple <code>(c, args)</code> where self is written as a Mul, <code>m</code>.</td>
</tr>
<tr>
<td><code>as_coeff_terms(*deps)</code></td>
<td>This method is deprecated.</td>
</tr>
<tr>
<td><code>as_coefficient(expr)</code></td>
<td>Extracts symbolic coefficient at the given expression.</td>
</tr>
<tr>
<td><code>as_coefficients_dict()</code></td>
<td></td>
</tr>
<tr>
<td><code>as_content_primitive(*radical)</code></td>
<td>This method should recursively remove a Rational from all arguments</td>
</tr>
<tr>
<td><code>as_dummy()</code></td>
<td></td>
</tr>
<tr>
<td><code>as_expr(*gens)</code></td>
<td>Convert a polynomial to a SymPy expression.</td>
</tr>
<tr>
<td><code>as_independent(*deps, **hint)</code></td>
<td>A mostly naive separation of a Mul or Add into arguments that are not</td>
</tr>
<tr>
<td><code>as_leading_term(*args, **kw_args)</code></td>
<td>Returns the leading (nonzero) term of the series expansion of self.</td>
</tr>
<tr>
<td><code>as_numer_denom()</code></td>
<td>expression -&gt; <code>a/b</code> -&gt; <code>a</code>, <code>b</code></td>
</tr>
<tr>
<td><code>as_ordered_factors(*order)</code></td>
<td>Return list of ordered factors (if Mul) else <code>[self]</code>.</td>
</tr>
<tr>
<td><code>as_ordered_terms(*order, data)</code></td>
<td>Transform an expression to an ordered list of terms.</td>
</tr>
<tr>
<td><code>as_poly(*gens, **args)</code></td>
<td>Converts self to a polynomial or returns None.</td>
</tr>
<tr>
<td><code>as_powers_dict()</code></td>
<td></td>
</tr>
<tr>
<td><code>as_real_imag()</code></td>
<td>Return self as a dictionary of factors with each factor being treated as a power.</td>
</tr>
<tr>
<td><code>as_terms()</code></td>
<td>Transform an expression to a list of terms.</td>
</tr>
<tr>
<td><code>atoms(*types)</code></td>
<td>Returns the atoms that form the current object.</td>
</tr>
<tr>
<td><code>cancel(*gens, **args)</code></td>
<td>See the <code>cancel</code> function in sympy.polys</td>
</tr>
<tr>
<td><code>class_key()</code></td>
<td></td>
</tr>
<tr>
<td><code>coeff(x, n, right)</code></td>
<td>Returns the coefficient from the term containing “x**n” or None if</td>
</tr>
<tr>
<td><code>collect(*args, **kw_args)</code></td>
<td>See the <code>collect</code> function in sympy.simplify</td>
</tr>
<tr>
<td><code>combsimp()</code></td>
<td>See the <code>combsimp</code> function in sympy.simplify</td>
</tr>
<tr>
<td><code>compare(other)</code></td>
<td>Return <code>-1</code>, <code>0</code>, <code>1</code> if the object is smaller, equal, or greater than other.</td>
</tr>
<tr>
<td><code>compare_pretty(*args, **kwargs)</code></td>
<td>Is <code>a &gt; b</code> in the sense of ordering in printing?</td>
</tr>
<tr>
<td><code>compute_leading_term(x, skip_abs, logx)</code></td>
<td>as_leading_term is only allowed for results of <code>.series()</code></td>
</tr>
<tr>
<td><code>conjugate()</code></td>
<td></td>
</tr>
<tr>
<td><code>copy()</code></td>
<td></td>
</tr>
<tr>
<td><code>could_extract_minus_sign()</code></td>
<td>Canonical way to choose an element in the set <code>{e, -e}</code> where <code>e</code> is any expression.</td>
</tr>
<tr>
<td><code>count(query)</code></td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td><code>count_ops(*visual)</code></td>
<td>wrapper for <code>count_ops</code> that returns the operation count.</td>
</tr>
<tr>
<td><code>diff(*symbols, **assumptions)</code></td>
<td></td>
</tr>
<tr>
<td><code>doit(**hints)</code></td>
<td></td>
</tr>
<tr>
<td><code>dummy_eq(other[, symbol])</code></td>
<td>Compare two expressions and handle dummy symbols.</td>
</tr>
<tr>
<td><code>equals(other[, failing_expression])</code></td>
<td>Return True if <code>self == other</code>, False if it doesn’t, or None. If</td>
</tr>
<tr>
<td><code>evalf(*args, **kw_args)</code></td>
<td>Expand an expression using hints.</td>
</tr>
<tr>
<td><code>extract_additively(c)</code></td>
<td>Return self - <code>c</code> if it’s possible to subtract <code>c</code> from self and</td>
</tr>
<tr>
<td><code>extract_branch_factor(*allow_half)</code></td>
<td>Try to write self as <code>exp_polar(2*pi*I*n)*z</code> in a nice way.</td>
</tr>
<tr>
<td><code>extract_multiplicatively(c)</code></td>
<td>Return None if it’s not possible to make self in the form</td>
</tr>
<tr>
<td><code>factor(*gens, **args)</code></td>
<td>See the <code>factor()</code> function in sympy.polys.polytools</td>
</tr>
<tr>
<td><code>find(query[, group])</code></td>
<td>Find all subexpressions matching a query.</td>
</tr>
</tbody>
</table>

Table 52.3 – continued from previous page
Table 52.3 – continued from previous page

<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fromiter(args, **assumptions)</td>
<td>Create a new object from an iterable.</td>
</tr>
<tr>
<td>getO()</td>
<td>Returns the additive O(..) symbol if there is one, else None.</td>
</tr>
<tr>
<td>getn()</td>
<td>Returns the order of the expression.</td>
</tr>
<tr>
<td>has(*args, **kw_args)</td>
<td>Test whether any subexpression matches any of the patterns.</td>
</tr>
<tr>
<td>integrate(*args, **kwargs)</td>
<td>See the integrate function in sympy.integrals</td>
</tr>
<tr>
<td>invert(g)</td>
<td>See the invert function in sympy.polys</td>
</tr>
<tr>
<td>is_constant(*wrt, **flags)</td>
<td></td>
</tr>
<tr>
<td>is_hypergeometric(k)</td>
<td></td>
</tr>
<tr>
<td>is_polynomial(*syms)</td>
<td>Return True if self is a polynomial in syms and False otherwise.</td>
</tr>
<tr>
<td>is_rational_function(*syms)</td>
<td>Test whether function is a ratio of two polynomials in the given symbols, syms.</td>
</tr>
<tr>
<td>iter_basic_args()</td>
<td>Iterates arguments of self.</td>
</tr>
<tr>
<td>leadterm(x)</td>
<td>Returns the leading term a^x**b as a tuple (a, b).</td>
</tr>
<tr>
<td>limit(x, xlim[, dir])</td>
<td>Compute limit x-&gt;xlim.</td>
</tr>
<tr>
<td>lseries(x, x0, dir)</td>
<td>Wrapper for series yielding an iterator of the terms of the series.</td>
</tr>
<tr>
<td>match(pattern)</td>
<td>Pattern matching.</td>
</tr>
<tr>
<td>matches(expr[, repl_dict])</td>
<td></td>
</tr>
<tr>
<td>n([n, subs, maxn, chop, strict, quad, verbose])</td>
<td>Evaluate the given formula to an accuracy of n digits.</td>
</tr>
<tr>
<td>normal()</td>
<td></td>
</tr>
<tr>
<td>nseries((x, x0, n, dir, logx))</td>
<td>Wrapper to _eval_nseries if assumptions allow, else to series.</td>
</tr>
<tr>
<td>nsimplify([constants, tolerance, full])</td>
<td>See the nsimplify function in sympy.simplify</td>
</tr>
<tr>
<td>powseries([deep, combine])</td>
<td>See the powsimp function in sympy.simplify</td>
</tr>
<tr>
<td>primitive()</td>
<td>Return the positive Rational that can be extracted non-recursively</td>
</tr>
<tr>
<td>radsimp()</td>
<td>See the radsimp function in sympy.simplify</td>
</tr>
<tr>
<td>ratsimp()</td>
<td>See the ratsimp function in sympy.simplify</td>
</tr>
<tr>
<td>refine((assumption))</td>
<td>See the refine function in sympy.assumptions</td>
</tr>
<tr>
<td>removeO()</td>
<td>Removes the additive O(..) symbol if there is one</td>
</tr>
<tr>
<td>rewrite(query, value[, map])</td>
<td>Replace matching subexpressions of self with value.</td>
</tr>
<tr>
<td>round([p])</td>
<td>Return x rounded to the given decimal place.</td>
</tr>
<tr>
<td>separate([deep, force])</td>
<td>See the separate function in sympy.simplify</td>
</tr>
<tr>
<td>series((x, x0, n, dir))</td>
<td>Series expansion of &quot;self&quot; around x = x0 yielding either terms of</td>
</tr>
<tr>
<td>simplify()</td>
<td>See the simplify function in sympy.simplify</td>
</tr>
<tr>
<td><strong>init</strong>()</td>
<td>x.<strong>init</strong>(...) initializes x; see help(type(x)) for signature</td>
</tr>
<tr>
<td>adjoint()</td>
<td></td>
</tr>
<tr>
<td>apart (x=None, **args)</td>
<td>See the apart function in sympy.polys</td>
</tr>
<tr>
<td>args</td>
<td>Returns a tuple of arguments of ‘self’.</td>
</tr>
</tbody>
</table>

52.2. Classes

359
Notes

Never use self._args, always use self.args. Only use _args in __new__ when creating a new function. Don’t override .args() from Basic (so that it’s easy to change the interface in the future if needed).

Examples

```python
>>> from sympy import symbols, cot
>>> from sympy.abc import x, y

>>> cot(x).args
(x,)

>>> cot(x).args[0]
x

>>> (x*y).args
(x, y)

>>> (x*y).args[1]
y
```

`args_cnc(cset=False, warn=True)`
Return [commutative factors, non-commutative factors] of self.

self is treated as a Mul and the ordering of the factors is maintained. If cset is True the commutative factors will be returned in a set. If there were repeated factors (as may happen with an unevaluated Mul) then an error will be raised unless it is explicitly suppressed by setting warn to False.

Note: -1 is always separated from a Number.

```python
>>> from sympy import symbols, oo
>>> A, B = symbols('A B', commutative=0)
>>> x, y = symbols('x y')

>>> (-2*x*y).args_cnc()
[[-1, 2, x, y], []]

>>> (-2.5*x).args_cnc()
[[-1, 2.5, x], []]

>>> (-2*x*A*B+y).args_cnc()
[[-1, 2, x, y], [A, B]]

>>> (-2*x*y).args_cnc(cset=True)
[set([[-1, 2, x, y]], [])]
```

The arg is always treated as a Mul:

```python
>>> (-2 + x + A).args_cnc()
[[], [x - 2 + A]]

>>> (-oo).args_cnc()  # -oo is a singleton
[[-1, oo], []]
```

`as_base_exp()`

`as_coeff_Add()`
Efficiently extract the coefficient of a summation.

`as_coeff_Mul(rational=False)`
Efficiently extract the coefficient of a product.
as_coeff_add(*deps)
Return the tuple (c, args) where self is written as an Add, a.
c should be a Rational added to any terms of the Add that are independent of deps.
args should be a tuple of all other terms of a; args is empty if self is a Number or if self is independent of
deps (when given).
This should be used when you don’t know if self is an Add or not but you want to treat self as an Add or
if you want to process the individual arguments of the tail of self as an Add.
• if you know self is an Add and want only the head, use self.args[0];
• if you don’t want to process the arguments of the tail but need the tail then use self.as_two_terms()
  which gives the head and tail.
• if you want to split self into an independent and dependent parts use
  self.as_independent(*deps)

>>> from sympy import *
>>> from sympy.abc import x, y
>>> (S(3)).as_coeff_add()
(3, ())
>>> (3 + x).as_coeff_add()
(3, (x,))
>>> (3 + x + y).as_coeff_add(x)
y + 3, (x,))
>>> (3 + y).as_coeff_add(x)
y + 3, ()

as_coeff_exponent(x)
c*x**e -> c,e where x can be any symbolic expression.

as_coeff_factors(*deps)
This method is deprecated. Use .as_coeff_add() instead.

as_coeff_mul(*deps)
Return the tuple (c, args) where self is written as a Mul, m.
c should be a Rational multiplied by any terms of the Mul that are independent of deps.
args should be a tuple of all other terms of m; args is empty if self is a Number or if self is independent of
deps (when given).
This should be used when you don’t know if self is a Mul or not but you want to treat self as a Mul or if
you want to process the individual arguments of the tail of self as a Mul.
• if you know self is a Mul and want only the head, use self.args[0];
• if you don’t want to process the arguments of the tail but need the tail then use self.as_two_terms()
  which gives the head and tail;
• if you want to split self into an independent and dependent parts use
  self.as_independent(*deps)

>>> from sympy import *
>>> from sympy.abc import x, y
>>> (S(3)).as_coeff_mul()
(3, ())
>>> (3*x*y).as_coeff_mul()
(3, (x, y))
>>> (3*x*y).as_coeff_mul(x)
(3*y, (x,))
>>> (3*y).as_coeff_mul(x)
(3*y, (x))

as_coeff_terms(*deps)
This method is deprecated. Use .as_coeff_mul() instead.

as_coefficient(expr)
Extracts symbolic coefficient at the given expression. In other words, this functions separates 'self' into
the product of 'expr' and 'expr'-free coefficient. If such separation is not possible it will return None.

See Also:
coeff

Examples

>>> from sympy import E, pi, sin, I, symbols
>>> from sympy.abc import x, y

>>> E.as_coefficient(E)
1
>>> (2*E).as_coefficient(E)
2
>>> (2*sin(E)*E).as_coefficient(E)
E
>>> (2*E + x*E).as_coefficient(E)
x + 2
>>> (2*E*x + x).as_coefficient(E)
E + x
>>> (E*(x + 1) + x).as_coefficient(E)
1
>>> (2*pi*I).as_coefficient(pi*I)
2
>>> (2*I).as_coefficient(pi*I)

as_coefficients_dict()
Return a dictionary mapping terms to their Rational coefficient. Since the dictionary is a defaultdict,
inquiries about terms which were not present will return a coefficient of 0. If an expression is not an Add
it is considered to have a single term.

Examples

>>> from sympy.abc import a, x

>>> (3*x + a*x + 4).as_coefficients_dict()
{1: 4, x: 3, a*x: 1}
>>> __[a]
0
>>> (3*a*x).as_coefficients_dict()
{a*x: 3}

as_content_primitive(radical=False)
This method should recursively remove a Rational from all arguments and return that
(content) and the new self (primitive). The content should always be positive and Mul(*foo.as_content_primitive()) == foo. The primitive need not be in canonical form and should try to preserve the underlying structure if possible (i.e. expand_mul should not be applied
to self).
Examples

```python
>>> from sympy import sqrt
>>> from sympy.abc import x, y, z

>>> eq = 2 + 2*x + 2*y*(3 + 3*y)

The as_contentPrimitive function is recursive and retains structure:

```python
>>> eq.as_contentPrimitive()
(2, x + 3*y*(y + 1) + 1)
```

Integer powers will have Rationals extracted from the base:

```python
>>> ((2 + 6*x)**2).as_contentPrimitive()
(4, (3*x + 1)**2)
>>> ((2 + 6*x)**(2*y)).as_contentPrimitive()
(1, (2*(3*x + 1))**(2*y))
```

Terms may end up joining once their as_contentPrimitives are added:

```python
>>> ((5*(x*(1 + y)) + 2*x*(3 + 3*y))).as_contentPrimitive()
(11, x*(y + 1))
>>> ((3*(x*(1 + y)) + 2*x*(3 + 3*y))).as_contentPrimitive()
(9, x*(y + 1))
>>> ((3*(z*(1 + y)) + 2.0*x*(3 + 3*y))).as_contentPrimitive()
(1, 2*(3*x + 1)**(2*y))
```

Radical content can also be factored out of the primitive:

```python
>>> (2*sqrt(2) + 4*sqrt(10)).as_contentPrimitive(radical=True)
(2, sqrt(2)*(1 + 2*sqrt(5)))
```

as_dummy()

as_expr(*gens)

Convert a polynomial to a SymPy expression.

Examples

```python
>>> from sympy import sin
>>> from sympy.abc import x, y

>>> f = (x**2 + x*y).as_poly(x, y)
>>> f.as_expr()
x**2 + x*y

>>> sin(x).as_expr()
sin(x)
```

as_independent(*deps, **hint)

A mostly naive separation of a Mul or Add into arguments that are not are dependent on deps. To obtain as complete a separation of variables as possible, use a separation method first, e.g.:
• `separatevars()` to change `Mul`, `Add` and `Pow` (including `exp`) into `Mul`
• `expand(mul=True)` to change `Add` or `Mul` into `Add`
• `expand(log=True)` to change `log expr` into an `Add`

The only non-naive thing that is done here is to respect noncommutative ordering of variables.
The returned tuple `(i, d)` has the following interpretation:
• `i` will have no variable that appears in `deps`
• `d` will be 1 or else have terms that contain variables that are in `deps`
  • if `self` is an `Add` then `self = i + d`
  • if `self` is a `Mul` then `self = i*d`
  • if `self` is anything else, either tuple `(self, S.One)` or `(S.One, self)` is returned.
To force the expression to be treated as an `Add`, use the hint `as_Add=True`

**Examples**

– `self` is an `Add`

```python
>>> from sympy import sin, cos, exp
>>> from sympy.abc import x, y, z

>>> (x + x*y).as_independent(x)
(0, x*y + x)
>>> (x + x*y).as_independent(y)
(x, x*y)
>>> (2*x*sin(x) + y + x + z).as_independent(x)
(y + z, 2*x*sin(x) + x)
>>> (2*x*sin(x) + y + x + z).as_independent(x, y)
(z, 2*x*sin(x) + x + y)
```

– `self` is a `Mul`

```python
>>> (x*sin(x)*cos(y)).as_independent(x)
(cos(y), x*sin(x))
```

Non-commutative terms cannot always be separated out when `self` is a `Mul`

```python
>>> from sympy import symbols
>>> n1, n2, n3 = symbols(’n1 n2 n3’, commutative=False)
>>> (n1 + n1*n2).as_independent(n2)
(n1, n1*n2)
>>> (n2*n1 + n1*n2).as_independent(n2)
(0, n1*n2 + n2*n1)
>>> (n1*n2+n3).as_independent(n1)
(1, n1*n2*n3)
>>> (n1*n2+n3).as_independent(n2)
(n1, n2*n3)
>>> ((x-n1)*(x-y)).as_independent(x)
(1, (x - y)*(x - n1))
```

– `self` is anything else:
```python
>>> (sin(x)).as_independent(x)
(1, sin(x))
>>> (sin(x)).as_independent(y)
(sin(x), 1)
>>> exp(x+y).as_independent(x)
(1, exp(x + y))

– force self to be treated as an Add:

>>> (3*x).as_independent(x, as_Add=True)
(0, 3*x)

– force self to be treated as a Mul:

>>> (3+x).as_independent(x, as_Add=False)
(1, x + 3)
>>> (-3+x).as_independent(x, as_Add=False)
(1, x - 3)

Note how the below differs from the above in making the constant on the dep term positive.

>>> (y*(-3+x)).as_independent(x)
(y, x - 3)

– use .as_independent() for true independence testing instead of .has(). The former considers only symbols in the free symbols while the latter considers all symbols

```from sympy import Integral
```
```python
>>> I = Integral(x, (x, 1, 2))
>>> I.has(x)
True
>>> x in I.free_symbols
False
>>> I.as_independent(x) == (I, 1)
True
>>> (I + x).as_independent(x) == (I, x)
True
```

Note: when trying to get independent terms, a separation method might need to be used first. In this case, it is important to keep track of what you send to this routine so you know how to interpret the returned values

```python
>>> from sympy import separatevars, log
>>> separatevars(exp(x+y)).as_independent(x)
(exp(y), exp(x))
>>> (x + x*y).as_independent(y)
(x, x*y)
>>> separatevars(x + x*y).as_independent(y)
(x, y + 1)
>>> (x*(1 + y)).as_independent(y)
(x, y + 1)
>>> (x*(1 + y)).expand(mul=True).as_independent(y)
(x, x*y)
```

```python
a, b=symbols('a b',positive=True)
>>> (log(a*b).expand(log=True)).as_independent(b)
(log(a), log(b))
```

See also: .separatevars(), .expand(log=True), .as_two_terms(), .as_coeff_add(), .as_coeff_mul()
as_leading_term(*args, **kw_args)
Returns the leading (nonzero) term of the series expansion of self.
The _eval_as_leading_term routines are used to do this, and they must always return a non-zero value.

Examples
>>> from sympy.abc import x
>>> (1 + x + x**2).as_leading_term(x)
1
>>> (1/x**2 + x + x**2).as_leading_term(x)
x**(-2)

as_numer_denom()
expression -> a/b -> a, b
This is just a stub that should be defined by an object’s class methods to get anything else.

See Also:
normal  return a/b instead of a, b

as_ordered_factors(order=None)
Return list of ordered factors (if Mul) else [self].

as_ordered_terms(order=None, data=False)
Transform an expression to an ordered list of terms.

Examples
>>> from sympy import sin, cos
>>> from sympy.abc import x, y

>>> (sin(x)**2*cos(x) + sin(x)**2 + 1).as_ordered_terms()
[sin(x)**2*cos(x), sin(x)**2, 1]

as_poly(*gens, **args)
Converts self to a polynomial or returns None.

Examples
>>> from sympy import Poly, sin
>>> from sympy.abc import x, y

>>> print (x**2 + x*y).as_poly()
Poly(x**2 + x*y, x, y, domain='ZZ')

>>> print (x**2 + x*y).as_poly(x, y)
Poly(x**2 + x*y, x, y, domain='ZZ')

>>> print (x**2 + sin(y)).as_poly(x, y)
None

as_powers_dict()
Return self as a dictionary of factors with each factor being treated as a power. The keys are the bases of the factors and the values, the corresponding exponents. The resulting dictionary should be used with caution if the expression is a Mul and contains non-commutative factors since the order that they appeared will be lost in the dictionary.
as_real_imag(\texttt{deep=True, **hints})

\texttt{as_terms()}

Transform an expression to a list of terms.

\texttt{assumptions0}

\texttt{atoms(*types)}

Returns the atoms that form the current object.

By default, only objects that are truly atomic and can’t be divided into smaller pieces are returned: symbols, numbers, and number symbols like I and pi. It is possible to request atoms of any type, however, as demonstrated below.

\textbf{Examples}

```python
>>> from sympy import Number, NumberSymbol, Symbol

>>> (1 + x + 2*sin(y + I*pi)).atoms(Symbol)
set([x, y])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Number)
set([1, 2])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Number, NumberSymbol)
set([1, 2, pi])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Number, NumberSymbol, I)
set([1, 2, I, pi])
```

Note that I (imaginary unit) and zoo (complex infinity) are special types of number symbols and are not part of the NumberSymbol class.

The type can be given implicitly, too:

```python
>>> (1 + x + 2*sin(y + I*pi)).atoms(x)  # x is a Symbol
set([x, y])
```

Be careful to check your assumptions when using the implicit option since \texttt{S(1).is_Integer = True} but \texttt{type(S(1))} is \texttt{One}, a special type of sympy atom, while \texttt{type(S(2))} is \texttt{type Integer} and will find all integers in an expression:

```python
>>> from sympy import S

>>> (1 + x + 2*sin(y + I*pi)).atoms(S(1))
set([1])

>>> (1 + x + 2*sin(y + I*pi)).atoms(S(2))
set([1, 2])
```

Finally, arguments to atoms() can select more than atomic atoms: any sympy type (loaded in core/__init__.py) can be listed as an argument and those types of “atoms” as found in scanning the arguments of the expression recursively:

```python
>>> from sympy import Function, Mul

>>> from sympy.core.function import AppliedUndef

>>> f = Function('f')

>>> (1 + f(x) + 2*sin(y + I*pi)).atoms(Function)
set([f(x), sin(y + I*pi)])

>>> (1 + f(x) + 2*sin(y + I*pi)).atoms(AppliedUndef)
set([f(x)])
```
>>> (1 + x + 2*sin(y + I*pi)).atoms(Mul)
set([I*pi, 2*sin(y + I*pi)])

cancel(*gens, **args)
    See the cancel function in sympy.polys

classmethod class_key()

coeff(x, n=1, right=False)
    Returns the coefficient from the term containing “x**n” or None if there is no such term. If
    n is zero then all terms independent of x will be returned.

    When x is noncommutative, the coeff to the left (default) or right of x can be returned. The keyword ‘right’
    is ignored when x is commutative.

    See Also:
    as_coeff_Add a method to separate the additive constant from an expression
    as_coeff_Mul a method to separate the multiplicative constant from an expression
    as_independent a method to separate x dependent terms/factors from others

Examples

>>> from sympy import symbols
>>> from sympy.abc import x, y, z

You can select terms that have an explicit negative in front of them:

>>> (-x + 2*y).coeff(-1)
x
>>> (x - 2*y).coeff(-1)
2*y

You can select terms with no Rational coefficient:

>>> (x + 2*y).coeff(1)
x
>>> (3 + 2*x + 4*x**2).coeff(1)
0

You can select terms independent of x by making n=0; in this case expr.as_independent(x)[0] is returned
(and 0 will be returned instead of None):

>>> (3 + 2*x + 4*x**2).coeff(x, 0)
3
>>> eq = ((x + 1)**3).expand() + 1
>>> eq
x**3 + 3*x**2 + 3*x + 2
>>> [eq.coeff(x, i) for i in reversed(range(4))]
[1, 3, 3, 2]
>>> eq -= 2
>>> [eq.coeff(x, i) for i in reversed(range(4))]
[1, 3, 3, 0]

You can select terms that have a numerical term in front of them:
```python
>>> (-x - 2*y).coeff(2)
-y
>>> from sympy import sqrt
>>> (x + sqrt(2)*x).coeff(sqrt(2))
x

The matching is exact:

```python
>>> (3 + 2*x + 4*x**2).coeff(x)
2
>>> (3 + 2*x + 4*x**2).coeff(x**2)
4
>>> (3 + 2*x + 4*x**2).coeff(x**3)
0
>>> (z*(x + y)**2).coeff((x + y)**2)
z
>>> (z*(x + y)**2).coeff(x + y)
0

In addition, no factoring is done, so 1 + z*(1 + y) is not obtained from the following:

```python
>>> (x + z*(x + x*y)).coeff(x)
1
```
If such factoring is desired, factor_terms can be used first:

```python
>>> from sympy import factor_terms
>>> factor_terms(x + z*(x + x*y)).coeff(x)
z*(y + 1) + 1
```

```python
n, m, o = symbols(’n m o’, commutative=False)
>>> n.coeff(n)
1
>>> (3*n).coeff(n)
3
>>> (n*m + m*n*m).coeff(n)  # = (1 + m)*n*m
1 + m
>>> (n*m + m*n*m).coeff(n, right=True)  # = (1 + m)*n*m
m
```
If there is more than one possible coefficient 0 is returned:

```python
>>> (n*m + m*n).coeff(n)
0
```
If there is only one possible coefficient, it is returned:

```python
>>> (n*m + x*m*n).coeff(m*n)
x
>>> (n*m + x*m*n).coeff(m*n, right=1)
1
```

```python
collect(syms, func=None, evaluate=True, exact=False, distribute_order_term=True)
```
See the collect function in sympy.simplify

```python
combsimp()
```
See the combsimp function in sympy.simplify

```python
compare(other)
```
Return -1, 0, 1 if the object is smaller, equal, or greater than other.
Not in the mathematical sense. If the object is of a different type from the “other” then their classes are ordered according to the sorted_classes list.

Examples

```python
>>> from sympy.abc import x, y
>>> x.compare(y)
-1
>>> x.compare(x)
0
>>> y.compare(x)
1
```

static `compare_pretty`(*args, **kwargs)
Is a > b in the sense of ordering in printing?

THIS FUNCTION IS DEPRECATED. Use `default_sort_key` instead.

yes ..... return 1
no ...... return -1
equal ... return 0

Strategy:

It uses Basic.compare as a fallback, but improves it in many cases, like \(x^{**3}, x^{**4}, O(x^{**3})\) etc. In those simple cases, it just parses the expression and returns the “sane” ordering such as:

\[1 < x < x^{**2} < x^{**3} < O(x^{**4})\] etc.

Examples

```python
>>> from sympy.abc import x
>>> from sympy import Basic, Number
>>> Basic._compare_pretty(x, x**2)
-1
>>> Basic._compare_pretty(x**2, x**2)
0
>>> Basic._compare_pretty(x**3, x**2)
1
>>> Basic._compare_pretty(Number(1, 2), Number(1, 3))
1
>>> Basic._compare_pretty(Number(0), Number(-1))
1
```

compute_leading_term\((x, skip_abs=False, logx=None)\)

as_leading_term is only allowed for results of .series() This is a wrapper to compute a series first. If skip_abs is true, the absolute term is assumed to be zero. (This is necessary because sometimes it cannot be simplified to zero without a lot of work, but is still known to be zero. See log._eval_nseries for an example.) If skip_log is true, log(x) is treated as an independent symbol. (This is needed for the gruntz algorithm.)

conjugate()

copy()

could_extract_minus_sign()

Canonical way to choose an element in the set \{e, -e\} where e is any expression. If the canonical element is e, we have e.could_extract_minus_sign() == True, else e.could_extract_minus_sign() == False.
For any expression, the set \{e.could_extract_minus_sign(), (-e).could_extract_minus_sign()\} must be \{True, False\}.

```python
>>> from sympy.abc import x, y
>>> (x-y).could_extract_minus_sign() != (y-x).could_extract_minus_sign()
True
```

count (query)
Count the number of matching subexpressions.

count_ops (visual=None)
wrapper for count_ops that returns the operation count.

default_assumptions = {}
diff (*symbols, **assumptions)
doit (**hints)
dummy_eq (other, symbol=None)
Compare two expressions and handle dummy symbols.

Examples

```python
>>> from sympy import Dummy
>>> from sympy.abc import x, y

>>> u = Dummy('u')

>>> (u**2 + 1).dummy_eq(x**2 + 1)
True
>>> (u**2 + 1) == (x**2 + 1)
False
>>> (u**2 + y).dummy_eq(x**2 + y, x)
True
>>> (u**2 + y).dummy_eq(x**2 + y, y)
False
```
equals (other, failing_expression=False)
Return True if self == other, False if it doesn’t, or None. If failing_expression is True then the expression which did not simplify to a 0 will be returned instead of None.

If self is a Number (or complex number) that is not zero, then the result is False.

If self is a number and has not evaluated to zero, evaf will be used to test whether the expression evaluates to zero. If it does so and the result has significance (i.e. the precision is either -1, for a Rational result, or is greater than 1) then the evaf value will be used to return True or False.

evalf (n=15, subs=None, maxn=100, chop=False, strict=False, quad=None, verbose=False)
Evaluate the given formula to an accuracy of n digits. Optional keyword arguments:

- `subs=dict` : Substitute numerical values for symbols, e.g. subs={x:3, y:1+pi}.
- `maxn=<integer>` : Allow a maximum temporary working precision of maxn digits (default=100)
- `chop=<bool>` : Replace tiny real or imaginary parts in subresults by exact zeros (default=False)
- `strict=<bool>` : Raise PrecisionExhausted if any subresult fails to evaluate to full accuracy, given the available maxprec (default=False)
quad=<str> Choose algorithm for numerical quadrature. By default, tanh-sinh quadrature is used. For oscillatory integrals on an infinite interval, try quad='osc'.

verbose=<bool> Print debug information (default=False)

expand(*args, **kw_args)

Expand an expression using hints.
See the docstring of the expand() function in sympy.core.function for more information.

extract_additively(c)
Return self - c if it’s possible to subtract c from self and make all matching coefficients move towards zero, else return None.

See Also:
extract_multiplicatively, coeff, as_coefficient

Examples

>>> from sympy import S
>>> from sympy.abc import x, y
>>> e = 2*x + 3
>>> e.extract_additively(x + 1)
(x + 2)
>>> e.extract_additively(3*x)
(x + 1)
>>> e.extract_additively(4)
4
>>> (y*(x + 1)).extract_additively(x + 1)
(y*(x + 1))
>>> ((x + 1)*(x + 2*y + 1) + 3).extract_additively(x + 1)
(x + 1)*(x + 2*y) + 3

Sometimes auto-expansion will return a less simplified result than desired; gcd_terms might be used in such cases:

>>> from sympy import gcd_terms
>>> (4*x*(y + 1) + y).extract_additively(x)
4*x*(y + 1) + x*(4*y + 3) - x*(4*y + 4) + y
>>> gcd_terms(_)
x*(4*y + 3) + y

extract_branch_factor(allow_half=False)

Try to write self as exp_polar(2*pi*I*n)*z in a nice way. Return (z, n).

>>> from sympy import exp_polar, I, pi
>>> from sympy.abc import x, y
>>> exp_polar(I*pi).extract_branch_factor()
(exp_polar(I*pi), 0)
>>> exp_polar(2*I*pi).extract_branch_factor()
(1, 1)
>>> exp_polar(-pi*I).extract_branch_factor()
(exp_polar(I*pi), -1)
>>> exp_polar(3*pi*I + x).extract_branch_factor()
(exp_polar(x + I*pi), 1)
>>> (y*exp_polar(-5*pi*I)*exp_polar(3*pi*I + 2*pi*x)).extract_branch_factor()
(y*exp_polar(2*pi*x), -1)
>>> exp_polar(-I*pi/2).extract_branch_factor()
(exp_polar(-I*pi/2), 0)

If allow_half is True, also extract exp_polar(I*pi):
Neuroimaging in Python Documentation, Release 0.3.0

```python
>>> exp_polar(I*pi).extract_branch_factor(allow_half=True)
(1, 1/2)
>>> exp_polar(2*I*pi).extract_branch_factor(allow_half=True)
(1, 1)
>>> exp_polar(3*I*pi).extract_branch_factor(allow_half=True)
(1, 3/2)
>>> exp_polar(-I*pi).extract_branch_factor(allow_half=True)
(1, -1/2)
```

**extract_multiplicatively** (*c*)

Return None if it’s not possible to make in the form c * something in a nice way, i.e. preserving the properties of arguments of self.

```python
>>> from sympy import symbols, Rational

>>> x, y = symbols('x,y', real=True)

>>> ((x*y)**3).extract_multiplicatively(x**2 * y)
x*y**2

>>> ((x*y)**3).extract_multiplicatively(x**4 * y)

>>> (2*x).extract_multiplicatively(2)
x

>>> (2*x).extract_multiplicatively(3)

>>> (Rational(1,2)*x).extract_multiplicatively(3)
x/6
```

**factor** (*gens, **args*)

See the factor() function in sympy.polys.polytools

**find** (*query, group=False*)

Find all subexpressions matching a query.

**formula**

Return a Formula with only terms=[self].

**free_symbols**

**classmethod fromiter** (*args, **assumptions*)

Create a new object from an iterable.

This is a convenience function that allows one to create objects from any iterable, without having to convert to a list or tuple first.

**Examples**

```python
>>> from sympy import Tuple

>>> Tuple.fromiter(i for i in xrange(5))
(0, 1, 2, 3, 4)
```

**func**

The top-level function in an expression.

The following should hold for all objects:
```python
>>> x == x.func(*x.args)

Examples

```from sympy.abc import x
>>> a = 2*x
>>> a.func
<class 'sympy.core.mul.Mul'>
>>> a.args
(2, x)
>>> a.func(*a.args)
2*x
>>> a == a.func(*a.args)
True

```getO()```
Returns the additive O(.,) symbol if there is one, else None.

```getn()```
Returns the order of the expression.
The order is determined either from the O(...) term. If there is no O(...) term, it returns None.

Examples

```from sympy import O
from sympy.abc import x
>>> (1 + x + O(x**2)).getn()
2
>>> (1 + x).getn()
```

```has(*args, **kw_args)```
Test whether any subexpression matches any of the patterns.

Examples

```from sympy import sin, S
from sympy.abc import x, y, z
>>> (x**2 + sin(x*y)).has(z)
False
>>> (x**2 + sin(x*y)).has(x, y, z)
True
>>> x.has(x)
True
```

Note that expr.has(*patterns) is exactly equivalent to any(expr.has(p) for p in patterns). In particular, False is returned when the list of patterns is empty.

```integrate(*args, **kwargs)```
See the integrate function in sympy.integrals
invert(g)
    See the invert function in sympy.polys

is_Add = False
is_AlgebraicNumber = False
is_Atom = True
is_Boolean = False
is_Derivative = False
is_Dummy = False
is_Equality = False
is_Float = False
is_Function = False
is_Integer = False
is_Matrix = False
is_Mul = False
is_Not = False
is_Number = False
is_NumberSymbol = False
is_Order = False
is_Piecewise = False
is_Poly = False
is_Pow = False
is_Rational = False
is_Real
    Deprecated alias for is_Float
is_Relational = False
is_Symbol = True
is_Wild = False
is_antihermitian
is_bounded
is_commutative
is_comparable = False
is_complex
is_composite
is_constant (*wrt, **flags)
is_even
is_finite
is_hermitian
is_hypergeometric\( (k) \)
is_imaginary
is_infinitesimal
is_integer
is_irrational
is_negative
is_noninteger
is_nonnegative
is_nonpositive
is_nonzero
is_number
is_odd
is_polar

is_polynomial\( (*\text{syms}) \)
Return True if self is a polynomial in syms and False otherwise.

This checks if self is an exact polynomial in syms. This function returns False for expressions that are “polynomials” with symbolic exponents. Thus, you should be able to apply polynomial algorithms to expressions for which this returns True, and Poly(expr, *syms) should work only if and only if expr.is_polynomial(*syms) returns True. The polynomial does not have to be in expanded form. If no symbols are given, all free symbols in the expression will be used.

This is not part of the assumptions system. You cannot do Symbol(’z’, polynomial=True).

Examples

```python
>>> from sympy import Symbol
>>> x = Symbol(’x’)
>>> ((x**2 + 1)**4).is_polynomial(x)
True
>>> ((x**2 + 1)**4).is_polynomial()
True
>>> (2**x + 1).is_polynomial(x)
False

>>> n = Symbol(’n’, nonnegative=True, integer=True)
>>> (x**n + 1).is_polynomial(x)
False
```

This function does not attempt any nontrivial simplifications that may result in an expression that does not appear to be a polynomial to become one.

```python
>>> from sympy import sqrt, factor, cancel
>>> y = Symbol(’y’, positive=True)
>>> a = sqrt(y**2 + 2*y + 1)
>>> a.is_polynomial(y)
False
>>> factor(a)
y + 1
```
>> factor(a).is_polynomial(y)
True

>>> b = (y**2 + 2*y + 1)/(y + 1)
>>> b.is_polynomial(y)
False
>>> cancel(b)
y + 1
>>> cancel(b).is_polynomial(y)
True

See also .is_rational_function()

is_positive

is_prime

is_rational

is_rational_function(*syms)

Test whether function is a ratio of two polynomials in the given symbols, syms. When syms is not given, all free symbols will be used. The rational function does not have to be in expanded or in any kind of canonical form.

This function returns False for expressions that are “rational functions” with symbolic exponents. Thus, you should be able to call .as_numer_denom() and apply polynomial algorithms to the result for expressions for which this returns True.

This is not part of the assumptions system. You cannot do Symbol(‘z’, rational_function=True).

Examples

>>> from sympy import Symbol, sin
>>> from sympy.abc import x, y

>>> (x/y).is_rational_function()
True

>>> (x**2).is_rational_function()
True

>>> (x/sin(y)).is_rational_function(y)
False

>>> n = Symbol(‘n’, integer=True)
>>> (x**n + 1).is_rational_function(x)
False

This function does not attempt any nontrivial simplifications that may result in an expression that does not appear to be a rational function to become one.

>>> from sympy import sqrt, factor, cancel
>>> y = Symbol(‘y’, positive=True)
>>> a = sqrt(y**2 + 2*y + 1)/y
>>> a.is_rational_function(y)
False
>>> factor(a)
(y + 1)/y
>>> factor(a).is_rational_function(y)
True

See also `is_rational_function()`.

**is_real**

**is_unbounded**

**is_zero**

**iter_basic_args()**

Iterates arguments of `self`.

**Examples**

```python
>>> from sympy.abc import x
>>> a = 2*x
>>> a.iter_basic_args()
<...iterator object at 0x...>
>>> list(a.iter_basic_args())
[2, x]
```

**leadterm(x)**

Returns the leading term a\times x^b as a tuple (a, b).

**Examples**

```python
>>> from sympy.abc import x
>>> (1+x+x**2).leadterm(x)
(1, 0)
>>> (1/x**2+x+x**2).leadterm(x)
(1, -2)
```

**limit(x, xlim, dir='+')**

Compute limit x->xlim.

**lseries(x=None, x0=0, dir='+')**

Wrapper for series yielding an iterator of the terms of the series.

Note: an infinite series will yield an infinite iterator. The following, for example, will never terminate. It will just keep printing terms of the \sin(x) series:

```python
for term in sin(x).lseries(x):
    print term
```

The advantage of `lseries()` over `nseries()` is that many times you are just interested in the next term in the series (i.e. the first term for example), but you don’t know how many you should ask for in `nseries()` using the “n” parameter.

See also `nseries()`.

**match(pattern)**

Pattern matching.

Wild symbols match all.

Return `None` when expression (self) does not match with pattern. Otherwise return a dictionary such that:
pattern.xreplace(self.match(pattern)) == self

Examples

```python
given_formula = (2*x)**2
matches = e.match(p*q**r)
matches[p] = 4
matches[q] = x
matches[r] = 2
result = (p*q**r).xreplace(e.match(p*q**r))
```

matches(expr, repl_dict={})

Evaluate the given formula to an accuracy of n digits. Optional keyword arguments:

- **subs=**<dict> Substitute numerical values for symbols, e.g. subs={x:3, y:1+pi}.
- **maxn=**<integer> Allow a maximum temporary working precision of maxn digits (default=100)
- **chop=**<bool> Replace tiny real or imaginary parts in subresults by exact zeros (default=False)
- **strict=**<bool> Raise PrecisionExhausted if any subresult fails to evaluate to full accuracy, given the available maxprec (default=False)
- **quad=**<str> Choose algorithm for numerical quadrature. By default, tanh-sinh quadrature is used. For oscillatory integrals on an infinite interval, try quad='osc'.
- **verbose=**<bool> Print debug information (default=False)

name

normal()

nseries(x=**None**, x0=0, n=6, dir=’+’, logx=**None**)

Wrapper to _eval_nseries if assumptions allow, else to series.

If x is given, x0 is 0, dir=’+’, and self has x, then _eval_nseries is called. This calculates “n” terms in the innermost expressions and then builds up the final series just by “cross-multiplying” everything out.

Advantage – it’s fast, because we don’t have to determine how many terms we need to calculate in advance.

Disadvantage – you may end up with less terms than you may have expected, but the O(x**n) term appended will always be correct and so the result, though perhaps shorter, will also be correct.

If any of those assumptions is not met, this is treated like a wrapper to series which will try harder to return the correct number of terms.

See also lseries().

nsimplify(constants=[], tolerance=**None**, full=**False**)

See the nsimplify function in sympy.simplify

52.2. Classes
powsimp \((\text{deep}=False, \text{combine}='all')\)

See the powsimp function in sympy.simplify

primitive()

Return the positive Rational that can be extracted non-recursively from every term of self (i.e., self is treated like an Add). This is like the as_coeff_Mul() method but primitive always extracts a positive Rational (never a negative or a Float).

Examples

>>> from sympy.abc import x
>>> (3*(x + 1)**2).primitive()
(3, (x + 1)**2)

a = (6*x + 2); a.primitive()
(2, 3*x + 1)

b = (x/2 + 3); b.primitive()
(1/2, x + 6)

>>> (a*b).primitive() == (1, a*b)
True

radsimp()

See the radsimp function in sympy.simplify

ratsimp()

See the ratsimp function in sympy.simplify

refine \((\text{assumption}=\text{True})\)

See the refine function in sympy.assumptions

removeO()

Removes the additive O(..) symbol if there is one

replace \((\text{query}, \text{value}, \text{map}=\text{False})\)

Replace matching subexpressions of self with value.

If map = True then also return the mapping \{old: new\} where old was a sub-expression found with query and new is the replacement value for it.

Traverses an expression tree and performs replacement of matching subexpressions from the bottom to the top of the tree. The list of possible combinations of queries and replacement values is listed below:

See Also:

subs substitution of subexpressions as defined by the objects themselves.

xreplace exact node replacement in expr tree; also capable of using matching rules

Examples

Initial setup

>>> from sympy import log, sin, cos, tan, Wild
>>> from sympy.abc import x, y

1.1. type -> type obj.replace(sin, tan)
>>> f.replace(sin, cos)
log(cos(x)) + tan(cos(x**2))

>>> sin(x).replace(sin, cos, map=True)
(cos(x), {sin(x): cos(x)})

1.2. type -> func  obj.replace(sin, lambda arg: ...)

>>> f.replace(sin, lambda arg: sin(2*arg))
log(sin(2*x)) + tan(sin(2*x**2))

2.1. expr -> expr  obj.replace(sin(a), tan(a))

>>> a = Wild('a')

>>> f.replace(sin(a), tan(a))
log(tan(x)) + tan(tan(x**2))

2.2. expr -> func  obj.replace(sin(a), lambda a: ...)

>>> f.replace(sin(a), cos(a))
log(cos(x)) + tan(cos(x**2))

3.1. func -> func  obj.replace(lambda expr: ..., lambda expr: ...)

>>> g = 2*sin(x**3)

>>> g.replace(lambda expr: expr.is_Number, lambda expr: expr**2)
4*sin(x**9)

**rewrite** (*args, **hints)**
Rewrites expression containing applications of functions of one kind in terms of functions of different kind. For example you can rewrite trigonometric functions as complex exponentials or combinatorial functions as gamma function.

As a pattern this function accepts a list of functions to to rewrite (instances of DefinedFunction class). As rule you can use string or a destination function instance (in this case rewrite() will use the str() function).

There is also possibility to pass hints on how to rewrite the given expressions. For now there is only one such hint defined called ‘deep’. When ‘deep’ is set to False it will forbid functions to rewrite their contents.

>>> from sympy import sin, exp, I

>>> from sympy.abc import x, y

>>> sin(x).rewrite(sin, exp)
-I*(exp(I*x) - exp(-I*x))/2

**round** (*p=0*)
Return x rounded to the given decimal place.

If a complex number would results, apply round to the real and imaginary components of the number.

**Notes**

Do not confuse the Python builtin function, round, with the SymPy method of the same name. The former always returns a float (or raises an error if applied to a complex value) while the latter returns either a Number or a complex number:
>>> isinstance(round(S(123), -2), Number)
False
>>> isinstance(S(123).round(-2), Number)
True
>>> isinstance((3+I).round(), Mul)
True
>>> isinstance((1 + 3*I).round(), Add)
True

Examples

>>> from sympy import pi, E, I, S, Add, Mul, Number

>>> S(10.5).round()
11.
>>> pi.round()
3.
>>> pi.round(2)
3.14

The round method has a chopping effect:

>>> (2*pi + I/10).round()
6.
>>> (pi/10 + 2*I).round()
2.*I
>>> (pi/10 + E*I).round(2)
0.31 + 2.72*I

separate (deep=False, force=False)

See the separate function in sympy.simplify

series (x=None, x0=0, n=6, dir=’+’)

Series expansion of “self” around \( x = x_0 \) yielding either terms of the series one by one (the lazy series given when \( n=None \), else all the terms at once when \( n \neq None \).

Note: when \( n \neq None \), if an \( O() \) term is returned then the \( x \) in the in it and the entire expression represents \( x - x_0 \), the displacement from \( x_0 \). (If there is no \( O() \) term then the series was exact and \( x \) has it’s normal meaning.) This is currently necessary since sympy’s \( O() \) can only represent terms at \( x_0=0 \). So instead of:

\[
\cos(x).series(x0=1, n=2) \rightarrow (1 - x)\sin(1) + \cos(1) + O((x - 1)^2)
\]

which graphically looks like this:

```
| . . .
| \ . .
-----------
| . . .
| \ . .
```

the following is returned instead:

\(-x\sin(1) + \cos(1) + O(x^2)\)

whose graph is this:
which is identical to \(\cos(x + 1).\) series\(\(n=2)\).

**Usage:** Returns the series expansion of “self” around the point \(x = x_0\) with respect to \(x\) up to \(O(x^n)\)
(default \(n = 6)\).

If \(x=\text{None}\) and \(\text{self}\) is univariate, the univariate symbol will be supplied, otherwise an error will be raised.

```python
generated code```

If \(n=\text{None}\) then an iterator of the series terms will be returned.

```python
generated code```

For \(\text{dir}=+\) (default) the series is calculated from the right and for \(\text{dir}=-\) the series from the left. For smooth functions this flag will not alter the results.

```python
generated code```

**simplify()**

See the simplify function in sympy.simplify

**sort_key(*args, **kw_args)**

**subs(*args, **kwargs)**

Substitutes old for new in an expression after sympifying args.

**args is either:**

- two arguments, e.g. `foo.subs(old, new)`
- one iterable argument, e.g. `foo.subs(iterable)`. The iterable may be
  - an iterable container with (old, new) pairs. In this case the replacements are processed in the order given with successive patterns possibly affecting replacements already made.
o a dict or set whose key/value items correspond to old/new pairs. In this case the old/new pairs will be sorted by op count and in case of a tie, by number of args and the default_sort_key. The resulting sorted list is then processed as an iterable container (see previous).

If the keyword simultaneous is True, the subexpressions will not be evaluated until all the substitutions have been made.

See Also:

replace replacement capable of doing wildcard-like matching, parsing of match, and conditional replacements

xreplace exact node replacement in expr tree; also capable of using matching rules

Examples

```python
>>> from sympy import pi, exp
>>> from sympy.abc import x, y

>>> (1 + x*y).subs(x, pi)
pi*y + 1

>>> (1 + x*y).subs({x: pi, y: 2})
1 + 2*pi

>>> (1 + x*y).subs([(x, pi), (y, 2)])
1 + 2*pi

>>> reps = [(y, x**2), (x, 2)]

>>> (x + y).subs(reps)
x**2 + 2

>>> (x**2 + x**4).subs(x**2, y)
y**2 + y

To replace only the x**2 but not the x**4, use xreplace:

>>> (x**2 + x**4).xreplace({x**2: y})
x**4 + y

To delay evaluation until all substitutions have been made, set the keyword simultaneous to True:

>>> (x/y).subs([(x, 0), (y, 0)])
0

>>> (x/y).subs([(x, 0), (y, 0)], simultaneous=True)
nan

This has the added feature of not allowing subsequent substitutions to affect those already made:

>>> ((x + y)/y).subs((x + y: y, y: x + y))
1

>>> ((x + y)/y).subs((x + y: y, y: x + y), simultaneous=True)
y/(x + y)

In order to obtain a canonical result, unordered iterables are sorted by count_op length, number of arguments and by the default_sort_key to break any ties. All other iterables are left unsorted.

```
Neuroimaging in Python Documentation, Release 0.3.0

```python
>>> A = (sqrt(sin(2*x)), a)
>>> B = (sin(2*x), b)
>>> C = (cos(2*x), c)
>>> D = (x, d)
>>> E = (exp(x), e)

>>> expr = sqrt(sin(2*x))*sin(exp(x)*x)*cos(2*x) + sin(2*x)

>>> expr.subs(dict([A, B, C, D, E]))

a*c*sin(d*e) + b
```

together (*args, **kwargs)
See the together function in sympy.polys

transpose ()

trigsimp (deep=False, recursive=False)
See the trigsimp function in sympy.simplify

xreplace (rule)

52.2.4 Formula

class nipy.algorithms.statistics.formula.formulae.Formula (seq, char='b')

Bases: object

A Formula is a model for a mean in a regression model.

It is often given by a sequence of sympy expressions, with the mean model being the sum of each term multiplied by a linear regression coefficient.

The expressions may depend on additional Symbol instances, giving a non-linear regression model.

Methods

design (input[, param, return_float, contrasts])
Construct the design matrix, and optional contrast matrices.

fromrec (rec[, keep, drop])
Construct Formula from recarray

subs (old, new)
Perform a sympy substitution on all terms in the Formula

__init__ (seq, char='b')

Parameters  seq : sequence of sympy.Basic

char : str, optional

coefficients for regression coefficient

coeffs

Coefficients in the linear regression formula.

design (input, param=None, return_float=False, contrasts=None)
Construct the design matrix, and optional contrast matrices.

Parameters  input : np.recarray

Recarray including fields needed to compute the Terms in getparams(self.design_expr).

param : None or np.recarray

52.2. Classes
Recarray including fields that are not Terms in getparams(self.design_expr)

**return_float**: bool, optional
If True, return a np.float array rather than a np.recarray

**contrasts**: None or dict, optional
Contrasts. The items in this dictionary should be (str, Formula) pairs where a contrast matrix is constructed for each Formula by evaluating its design at the same parameters as self.design. If not None, then the return_float is set to True.

**Returns**
- **des**: 2D array
design matrix
- **cmatrices**: dict, optional
  Dictionary with keys from contrasts input, and contrast matrices corresponding to des design matrix. Returned only if contrasts input is not None

**design_expr**

**dtype**
The dtype of the design matrix of the Formula.

**static fromrec**(rec, keep=[], drop=[])
Construct Formula from recarray
For fields with a string-dtype, it is assumed that these are qualitative regressors, i.e. Factors.

**Parameters**
- **rec**: recarray
  Recarray whose field names will be used to create a formula.
- **keep**: []
  Field names to explicitly keep, dropping all others.
- **drop**: []
  Field names to drop.

**mean**
Expression for the mean, expressed as a linear combination of terms, each with dummy variables in front.

**params**
The parameters in the Formula.

**subs**(old, new)
Perform a sympy substitution on all terms in the Formula

Returns a new instance of the same class

**Parameters**
- **old**: sympy.Basic
  The expression to be changed
- **new**: sympy.Basic
  The value to change it to.

**Returns**
- **newf**: Formula
Examples

```python
>>> s, t = [Term(l) for l in 'st']
>>> f, g = [sympy.Function(l) for l in 'fg']
>>> form = Formula([f(t), g(s)])
>>> newform = form.subs(g, sympy.Function('h'))
>>> newform.terms
array([f(t), h(s)], dtype=object)
```

```python
>>> form.terms
array([f(t), g(s)], dtype=object)
```

```python
terms
Terms in the linear regression formula.
```

52.2.5 RandomEffects

class nipy.algorithms.statistics.formula.formulae.RandomEffects(seq, sigma=None, char='e')

Bases: nipy.algorithms.statistics.formula.formulae.Formula

Covariance matrices for common random effects analyses.

Examples

Two subjects (here named 2 and 3):

```python
>>> subj = make_recarray([2,2,2,3,3], 's')
>>> subj_factor = Factor('s', [2,3])
```

By default the covariance matrix is symbolic. The display differs a little between sympy versions (hence we don’t check it in the doctests):

```python
>>> c = RandomEffects(subj_factor.terms)
>>> c.cov(subj)
array([[[s2_0, s2_0, s2_0, 0, 0],
        [s2_0, s2_0, s2_0, 0, 0],
        [s2_0, s2_0, s2_0, 0, 0],
        [0, 0, 0, s2_1, s2_1],
        [0, 0, 0, s2_1, s2_1]], dtype=object)
```

With a numeric `sigma`, you get a numeric array:

```python
>>> c = RandomEffects(subj_factor.terms, sigma=np.array([[4,1],[1,6]]))
>>> c.cov(subj)
array([[ 4., 4., 4., 1., 1.],
        [ 4., 4., 4., 1., 1.],
        [ 4., 4., 4., 1., 1.],
        [ 1., 1., 1., 6., 6.],
        [ 1., 1., 1., 6., 6.]]
```

Methods

```python
cov(term[, param]) Compute the covariance matrix for some given data.
```

Continued on next page
Table 52.5 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>design()</code></td>
<td>Construct the design matrix, and optional contrast matrices.</td>
</tr>
<tr>
<td><code>fromrec()</code></td>
<td>Construct Formula from recarray</td>
</tr>
<tr>
<td><code>subs()</code></td>
<td>Perform a sympy substitution on all terms in the Formula</td>
</tr>
</tbody>
</table>

```python
__init__ (seq, sigma=None, char='e')
Initialize random effects instance

Parameters
- **seq**: [sympy.Basic]
  - Initialize random effects instance
- **sigma**: ndarray
  - Covariance of the random effects. Defaults to a diagonal with entries for each random effect.
- **char**: character for regression coefficient

coeffs
Coefficients in the linear regression formula.

cov (term, param=None)
Compute the covariance matrix for some given data.

Parameters
- **term**: np.recarray
  - Recarray including fields corresponding to the Terms in getparams(self.design_expr).
- **param**: np.recarray
  - Recarray including fields that are not Terms in getparams(self.design_expr)

Returns
- **C**: ndarray
  - Covariance matrix implied by design and self.sigma.

design (input, param=None, return_float=False, contrasts=None)
Construct the design matrix, and optional contrast matrices.

Parameters
- **input**: np.recarray
  - Recarray including fields needed to compute the Terms in getparams(self.design_expr).
- **param**: None or np.recarray
  - Recarray including fields that are not Terms in getparams(self.design_expr)
- **return_float**: bool, optional
  - If True, return a np.float array rather than a np.recarray
- **contrasts**: None or dict, optional
  - Contrasts. The items in this dictionary should be (str, Formula) pairs where a contrast matrix is constructed for each Formula by evaluating its design at the same parameters as self.design. If not None, then the return_float is set to True.

Returns
- **des**: 2D array
  - design matrix
- **cmatrices**: dict, optional
  - Dictionary with keys from contrasts input, and contrast matrices corresponding to des design matrix. Returned only if contrasts input is not None
```

388 Chapter 52. algorithms.statistics.formula.formulae
dtype
The dtype of the design matrix of the Formula.

static fromrec (rec, keep=[], drop=[])
Construct Formula from recarray

For fields with a string-dtype, it is assumed that these are qualitative regressors, i.e. Factors.

Parameters  rec: recarray :
Recarray whose field names will be used to create a formula.

keep: [] :
Field names to explicitly keep, dropping all others.

drop: [] :
Field names to drop.

mean
Expression for the mean, expressed as a linear combination of terms, each with dummy variables in front.

params
The parameters in the Formula.

subs (old, new)
Perform a sympy substitution on all terms in the Formula

Returns a new instance of the same class

Parameters  old : sympy.Basic
The expression to be changed

new : sympy.Basic
The value to change it to.

Returns  newf : Formula

Examples

```python
>>> s, t = [Term(l) for l in 'st']
>>> f, g = [sympy.Function(l) for l in 'fg']
>>> form = Formula([f(t), g(s)])
>>> newform = form.subs(g, sympy.Function('h'))
>>> newform.terms
array([f(t), h(s)], dtype=object)
>>> form.terms
array([f(t), g(s)], dtype=object)
```

terms
Terms in the linear regression formula.

52.2.6 Term

class nipy.algorithms.statistics.formula.formulae.Term
    Bases: sympy.core.symbol.Symbol

    A sympy.Symbol type to represent a term in a regression model
Terms can be added to other sympy expressions with the single convention that a term plus itself returns itself.

It is meant to emulate something on the right hand side of a formula in R. In particular, its name can be the name of a field in a recarray used to create a design matrix.

```python
>>> t = Term('x')
>>> xval = np.array([[3], [4], [5]], np.dtype({'x': np.float}))
>>> f = t.formula
>>> d = f.design(xval)
>>> print d.dtype.descr
[('x', '<f8')]
>>> f.design(xval, return_float=True)
array([ 3., 4., 5.])
```

**Methods**

- **__call__(*args)**
  - See the call function in sympy.polys
- **adjoint()**
- **apart([x])**
  - See the apart function in sympy.polys
- **args_cnc([cset, warn])**
  - Return [commutative factors, non-commutative factors] of self.
- **as_base_exp()**
- **as_coeff_Add()**
  - Efficiently extract the coefficient of a summation.
- **as_coeff_Mul([rational])**
  - Efficiently extract the coefficient of a product.
- **as_coeff_add([deps])**
  - Return the tuple (c, args) where self is written as an Add, a.
- **as_coeff_exponent(x)**
  - $c \times x^{\ast e} \rightarrow c, e$ where x can be any symbolic expression.
- **as_coeff_factors([deps])**
  - This method is deprecated.
- **as_coeff_mul([deps])**
  - Return the tuple (c, args) where self is written as a Mul, m.
- **as_coeff_terms([deps])**
  - This method is deprecated.
- **as_coefficient(expre)**
  - Extracts symbolic coefficient at the given expression.
- **as_coefficients_dict()**
  - Return a dictionary mapping terms to their Rational coefficient.
- **as_content_primitive([radical])**
  - This method should recursively remove a Rational from all arguments
- **as_dummy()**
- **as_expr(*gens)**
  - Convert a polynomial to a SymPy expression.
- **as_independent([deps, **hint])**
  - A mostly naive separation of a Mul or Add into arguments that are not
- **as_leading_term(*args, **kw_args)**
  - Returns the leading (nonzero) term of the series expansion of self.
- **as_numer_denom()**
  - expression $\rightarrow a/b -> a, b$
- **as_ordered_factors([order])**
  - Return list of ordered factors (if Mul) else [self].
- **as_ordered_terms([order, data])**
  - Transform an expression to an ordered list of terms.
- **as_poly(*gens, **args)**
  - Converts self to a polynomial or returns None.
- **as_powers_dict()**
  - Return self as a dictionary of factors with each factor being treated as a power.
- **as_real_imag([deep])**
- **as_terms()**
  - Transform an expression to a list of terms.
- **atoms(*types)**
  - Returns the atoms that form the current object.
- **cancel(*gens, **args)**
  - See the cancel function in sympy.polys
- **class_key()**
- **coeff(x, n, right)**
  - Returns the coefficient from the term containing “x**n” or None if
- **collect(syms[, func, evaluate, exact, ...])**
  - See the collect function in sympy.simplify
- **combsimp()**
  - See the combsimp function in sympy.simplify
- **compare(other)**
  - Return -1, 0, 1 if the object is smaller, equal, or greater than other.
- **compare_pretty(*args, **kwargs)**
  - Is a > b in the sense of ordering in printing?
- **compute_leading_term(x[, skip_abs, logx])**
  - as_leading_term is only allowed for results of .series()
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>conjugate()</td>
<td>Canonical way to choose an element in the set {e, -e} where ( e ) is any expression.</td>
</tr>
<tr>
<td>copy()</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>could_extract_minus_sign()</td>
<td>wrapper for count_ops that returns the operation count.</td>
</tr>
<tr>
<td>count(query)</td>
<td>Canonical way to choose an element in the set {e, -e} where ( e ) is any expression.</td>
</tr>
<tr>
<td>count_ops([visual])</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>diff(*symbols, **assumptions)</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>doit(**hints)</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>dummy_eq(other[, symbol])</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>equals(other[, failing_expression])</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>evalf([n, subs, maxn, chop, strict, quad, ...])</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>expand(*args, **kw_args)</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>extract_additively(c)</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>extract_branch_factor([allow_half])</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>extract_multiplicatively(c)</td>
<td>Count the number of matching subexpressions.</td>
</tr>
<tr>
<td>factor(*gens, **args)</td>
<td>See the factor() function in sympy.polys.polytools.</td>
</tr>
<tr>
<td>find(query[, group])</td>
<td>Find all subexpressions matching a query.</td>
</tr>
<tr>
<td>fromiter(args, **assumptions)</td>
<td>Create a new object from an iterable.</td>
</tr>
<tr>
<td>getO()</td>
<td>Returns the additive O(( _ )) symbol if there is one, else None.</td>
</tr>
<tr>
<td>getn()</td>
<td>Returns the order of the expression.</td>
</tr>
<tr>
<td>has(*args, **kw_args)</td>
<td>Test whether any subexpression matches any of the patterns.</td>
</tr>
<tr>
<td>integrate(*args, **kwargs)</td>
<td>See the integrate function in sympy.integrals</td>
</tr>
<tr>
<td>invert(g)</td>
<td>See the invert function in sympy.polys.</td>
</tr>
<tr>
<td>is_constant(*wrt, **flags)</td>
<td>Return True if self is a polynomial in syms and False otherwise.</td>
</tr>
<tr>
<td>is_hypergeometric(k)</td>
<td>Test whether function is a ratio of two polynomials in the given symbols, syms.</td>
</tr>
<tr>
<td>is_rational_function(*syms)</td>
<td>Iterates arguments of self.</td>
</tr>
<tr>
<td>leadterm(x)</td>
<td>Returns the leading term ( a x^n ) as a tuple ( (a, b) ).</td>
</tr>
<tr>
<td>limit(x, xlim[, dir])</td>
<td>Compute limit ( x \rightarrow \text{lim} ).</td>
</tr>
<tr>
<td>lseries([x, x0, dir])</td>
<td>Wrapper for series yielding an iterator of the terms of the series.</td>
</tr>
<tr>
<td>match(pattern)</td>
<td>Pattern matching.</td>
</tr>
<tr>
<td>matches(expr, repl_dict)</td>
<td>Evaluate the given formula to an accuracy of ( n ) digits.</td>
</tr>
<tr>
<td>n([n, subs, maxn, chop, strict, quad, verbose])</td>
<td>Evaluate the given formula to an accuracy of ( n ) digits.</td>
</tr>
<tr>
<td>normal()</td>
<td>See the normal() function in sympy.polys.polytools.</td>
</tr>
<tr>
<td>nseries(x, x0, n, dir, logx)</td>
<td>Wrapper to _eval_nseries if assumptions allow, else to series.</td>
</tr>
<tr>
<td>ncancel()</td>
<td>See the ncancel() function in sympy.polys.polytools.</td>
</tr>
<tr>
<td>cancel()</td>
<td>See the cancel() function in sympy.polys.polytools.</td>
</tr>
<tr>
<td>primitive()</td>
<td>Return the positive Rational that can be extracted non-recursively</td>
</tr>
<tr>
<td>radexpand()</td>
<td>See the radexpand() function in sympy.polys.polytools.</td>
</tr>
<tr>
<td>radsimp()</td>
<td>See the radsimp() function in sympy.polys.polytools.</td>
</tr>
<tr>
<td>ratsimp()</td>
<td>See the ratsimp() function in sympy.polys.polytools.</td>
</tr>
<tr>
<td>refine((assumption))</td>
<td>See the refine function in sympy.assumptions.</td>
</tr>
<tr>
<td>removeO()</td>
<td>Removes the additive O(( _ )) symbol if there is one</td>
</tr>
<tr>
<td>replace(query, value[, map])</td>
<td>Replace matching subexpressions of self with value.</td>
</tr>
<tr>
<td>Rewrite(*args, **hints)</td>
<td>Rewrites expression containing applications of functions of one kind in terms of</td>
</tr>
<tr>
<td>round(p)</td>
<td>Return ( x ) rounded to the given decimal place.</td>
</tr>
<tr>
<td>separate([deep, force])</td>
<td>See the separate function in sympy.polys.polytools.</td>
</tr>
<tr>
<td>series([x, x0, n, dir])</td>
<td>Series expansion of (&quot;self&quot;) around ( x = x0 ) yielding either terms of</td>
</tr>
<tr>
<td>simplify()</td>
<td>See the simplify function in sympy.polys.polytools.</td>
</tr>
<tr>
<td>sort_key(*args, **kw_args)</td>
<td>Substitutes old for new in an expression after sympifying args.</td>
</tr>
<tr>
<td>subs(*args, **kwargs)</td>
<td>Substitutes old for new in an expression after sympifying args.</td>
</tr>
</tbody>
</table>
__init__()
    x.__init__(...) initializes x; see help(type(x)) for signature

adjoint()

apart (x=None, **args)
    See the apart function in sympy.polys

args
    Returns a tuple of arguments of ‘self’.

Notes

Never use self._args, always use self.args. Only use _args in __new__ when creating a new function. Don’t override .args() from Basic (so that it’s easy to change the interface in the future if needed).

Examples

>>> from sympy import symbols, cot
>>> from sympy.abc import x, y

>>> cot(x).args
(x,)

>>> cot(x).args[0]
x

>>> (x+y).args
(x, y)

>>> (x+y).args[1]
y

args_cnc (cset=False, warn=True)
    Return [commutative factors, non-commutative factors] of self.

    self is treated as a Mul and the ordering of the factors is maintained. If cset is True the commutative factors will be returned in a set. If there were repeated factors (as may happen with an unevaluated Mul) then an error will be raised unless it is explicitly supressed by setting warn to False.

Note: -1 is always separated from a Number.

>>> from sympy import symbols, oo
>>> A, B = symbols('A B', commutative=0)
>>> x, y = symbols('x y')

>>> (-2*x+y).args_cnc()
[[-1, 2, x, y], []]

>>> (-2.5*x).args_cnc()
[[[-1, 2.5, x], []]]

>>> (-2*x+A*B+y).args_cnc()
The arg is always treated as a Mul:

>>> (-2 + x + A).args_cnc()
[[], [x - 2 + A]]

>>> (-oo).args_cnc()  # -oo is a singleton
[[-1, oo], []]

as_base_exp()

as_coeff_Add()
Efficiently extract the coefficient of a summation.

as_coeff_Mul(rational=False)
Efficiently extract the coefficient of a product.

as_coeff_add(*deps)

Return the tuple (c, args) where self is written as an Add, a.

  c should be a Rational added to any terms of the Add that are independent of deps.

  args should be a tuple of all other terms of a; args is empty if self is a Number or if self is independent of
deps (when given).

This should be used when you don’t know if self is an Add or not but you want to treat self as an Add or
if you want to process the individual arguments of the tail of self as an Add.

  • if you know self is an Add and want only the head, use self.args[0];

  • if you don’t want to process the arguments of the tail but need the tail then use self.as_two_terms()
which gives the head and tail.

  • if you want to split self into an independent and dependent parts use
self.as_independent(*deps)

>>> from sympy import S
>>> from sympy.abc import x, y

>>> (S(3)).as_coeff_add()
(3, ())

>>> (3 + x).as_coeff_add()
(3, (x,))

>>> (3 + x + y).as_coeff_add(x)
(y + 3, (x,))

as_coeff_exponent(x)

  c*x**e -> c, e where x can be any symbolic expression.

as_coeff_factors(*deps)
This method is deprecated. Use .as_coeff_add() instead.

as_coeff_mul(*deps)

Return the tuple (c, args) where self is written as a Mul, m.

  c should be a Rational multiplied by any terms of the Mul that are independent of deps.

  args should be a tuple of all other terms of m; args is empty if self is a Number or if self is independent of
deps (when given).
This should be used when you don’t know if self is a Mul or not but you want to treat self as a Mul or if you want to process the individual arguments of the tail of self as a Mul.

•if you know self is a Mul and want only the head, use self.args[0];
•if you don’t want to process the arguments of the tail but need the tail then use self.as_two_terms() which gives the head and tail;
•if you want to split self into an independent and dependent parts use self.as_independent(*deps)

```python
>>> from sympy import S
>>> from sympy.abc import x, y

>>> (S(3)).as_coeff_mul()
(3, ())
>>> (3*x*y).as_coeff_mul()
(3, (x, y))
>>> (3*x*y).as_coeff_mul(x)
(3*y, (x,))
>>> (3*y).as_coeff_mul(x)
(3*y, ())
```

as_coeff_terms(*deps)
This method is deprecated. Use .as_coeff_mul() instead.

as_coefficient(expr)
Extracts symbolic coefficient at the given expression. In other words, this functions separates ‘self’ into the product of ‘expr’ and ‘expr’-free coefficient. If such separation is not possible it will return None.

See Also:
coeff

Examples

```python
>>> from sympy import E, pi, sin, I, symbols
>>> from sympy.abc import x, y

>>> E.as_coefficient(E)
1
>>> (2*E).as_coefficient(E)
2
>>> (2*sin(E)*E).as_coefficient(E)

>>> (2*E + x*E).as_coefficient(E)
x + 2
>>> (2*E*x + x).as_coefficient(E)

>>> (E*(x + 1) + x).as_coefficient(E)

>>> (2*pi*I).as_coefficient(pi*I)
2
>>> (2*I).as_coefficient(pi*I)
```

as_coefficients_dict()
Return a dictionary mapping terms to their Rational coefficient. Since the dictionary is a defaultdict, inquiries about terms which were not present will return a coefficient of 0. If an expression is not an Add it is considered to have a single term.
Examples

```python
>>> from sympy.abc import a, x
>>> (3*x + a*x + 4).as_coefficients_dict()
{1: 4, x: 3, a*x: 1}
>>> _[a]
0
>>> (3*a*x).as_coefficients_dict()
{a*x: 3}
```

as_content_primitive (radical=False)

This method should recursively remove a Rational from all arguments and return that (content) and the new self (primitive). The content should always be positive and `Mul(*foo.as_content_primitive()) == foo`. The primitive need not be in canonical form and should try to preserve the underlying structure if possible (i.e. `expand_mul` should not be applied to self).

Examples

```python
>>> from sympy import sqrt
>>> from sympy.abc import x, y, z

>>> eq = 2 + 2*x + 2*y*(3 + 3*y)

The as_content_primitive function is recursive and retains structure:

```python
>>> eq.as_content_primitive()
(2, x + 3*y*(y + 1) + 1)
```

Integer powers will have Rationals extracted from the base:

```python
>>> ((2 + 6*x)**2).as_content_primitive()
(4, (3*x + 1)**2)
>>> ((2 + 6*x)**(2*y)).as_content_primitive()
(1, (2*(3*x + 1))**(2*y))
```

Terms may end up joining once their as_content_primitives are added:

```python
>>> ((5*(x*(1 + y)) + 2*x*(3 + 3*y))).as_content_primitive()
(11, x*(y + 1))
>>> ((3*(x*(1 + y)) + 2*x*(3 + 3*y))).as_content_primitive()
(9, x*(y + 1))
>>> ((3*(z*(1 + y)) + 2.0*x*(3 + 3*y))).as_content_primitive()
(9, 6.0*x*(y + 1) + 3*z*(y + 1))
>>> ((5*(x*(1 + y)) + 2.0*x*(3 + 3*y))**2).as_content_primitive()
(121, x**2*(y + 1)**2)
>>> ((5*(x*(1 + y)) + 2.0*x*(3 + 3*y))**2).as_content_primitive()
(1, 121.0*x**2*(y + 1)**2)
```

Radical content can also be factored out of the primitive:

```python
>>> (2*sqrt(2) + 4*sqrt(10)).as_content_primitive(radical=True)
(2, sqrt(2)*(1 + 2*sqrt(5)))
```

as_dummy ()

as_expr (*gens)

Convert a polynomial to a SymPy expression.
Examples

```python
>>> from sympy import sin
>>> from sympy.abc import x, y

>>> f = (x**2 + x*y).as_poly(x, y)
>>> f.as_expr()
x**2 + x*y

>>> sin(x).as_expr()
sin(x)
```

`as_independent(*deps, **hint)`
A mostly naive separation of a Mul or Add into arguments that are not are dependent on deps. To obtain as complete a separation of variables as possible, use a separation method first, e.g.:

- `separatevars()` to change Mul, Add and Pow (including exp) into Mul
- `.expand(mul=True)` to change Add or Mul into Add
- `.expand(log=True)` to change log expr into an Add

The only non-naive thing that is done here is to respect noncommutative ordering of variables.

The returned tuple `(i, d)` has the following interpretation:

- `i` will has no variable that appears in deps
- `d` will be 1 or else have terms that contain variables that are in deps
- if `self` is an Add then `self = i + d`
- if `self` is a Mul then `self = i*d`
- if `self` is anything else, either tuple `(self, S.One)` or `(S.One, self)` is returned.

To force the expression to be treated as an Add, use the hint `as_Add=True`

Examples

- self is an Add

```python
>>> from sympy import sin, cos, exp
>>> from sympy.abc import x, y, z

>>> (x + x*y).as_independent(x)
(0, x*y + x)

>>> (x + x*y).as_independent(y)
(x, x*y)

>>> (2*x*sin(x) + y + x + z).as_independent(x)
y + z, 2*x*sin(x) + x)

>>> (2*x*sin(x) + y + x + z).as_independent(x, y)
(z, 2*x*sin(x) + x + y)
```

- self is a Mul

```python
>>> (x*sin(x)*cos(y)).as_independent(x)
(cos(y), x*sin(x))
```

non-commutative terms cannot always be separated out when self is a Mul
>>> from sympy import symbols
>>> n1, n2, n3 = symbols('n1 n2 n3', commutative=False)
>>> (n1 + n1*n2).as_independent(n2)
(n1, n1*n2)
>>> (n2*n1 + n1*n2).as_independent(n2)
(0, n1*n2 + n2*n1)
>>> (n1*n2+n3).as_independent(n1)
(1, n1*n2*n3)
>>> (n1*n2+n3).as_independent(n2)
(n1, n2*n3)
>>> ((x-n1)*(x-y)).as_independent(x)
(1, (x - y)*(x - n1))

– self is anything else:

>>> (sin(x)).as_independent(x)
(1, sin(x))
>>> (sin(x)).as_independent(y)
(sin(x), 1)
>>> exp(x+y).as_independent(x)
(1, exp(x + y))

– force self to be treated as an Add:

>>> (3*x).as_independent(x, as_Add=True)
(0, 3*x)

– force self to be treated as a Mul:

>>> (3+x).as_independent(x, as_Add=False)
(1, x + 3)
>>> (-3+x).as_independent(x, as_Add=False)
(1, x - 3)

Note how the below differs from the above in making the constant on the dep term positive.

>>> (y*(-3+x)).as_independent(x)
(y, x - 3)

– use .as_independent() for true independence testing instead of .has(). The former considers only symbols in the free symbols while the latter considers all symbols

>>> from sympy import Integral
>>> I = Integral(x, (x, 1, 2))
>>> I.has(x)
True
>>> x in I.free_symbols
False
>>> I.as_independent(x) == (I, 1)
True
>>> (I + x).as_independent(x) == (I, x)
True

Note: when trying to get independent terms, a separation method might need to be used first. In this case, it is important to keep track of what you send to this routine so you know how to interpret the returned values

>>> from sympy import separatevars, log
>>> separatevars(exp(x+y)).as_independent(x)
(exp(y), exp(x))
>>> (x + x*y).as_independent(y)
(x, x*y)
>>> separatevars(x + x*y).as_independent(y)
(x, y + 1)
>>> (x*(1 + y)).as_independent(y)
(x, y + 1)
>>> (x*(1 + y)).expand(mul=True).as_independent(y)
(x, x*y)
>>> a, b=symbols('a b', positive=True)
>>> (log(a*b).expand(log=True)).as_independent(b)
(log(a), log(b))

See also: .separatevars(), .expand(log=True), .as_two_terms(), .as_coeff_add(), .as_coeff_mul()

as_leading_term(*args, **kw_args)
Returns the leading (nonzero) term of the series expansion of self.
The _eval_as_leading_term routines are used to do this, and they must always return a non-zero value.

Examples

>>> from sympy.abc import x
>>> (1 + x + x**2).as_leading_term(x)
1
>>> (1/x**2 + x + x**2).as_leading_term(x)
x**(-2)

as_numer_denom()
expression -> a/b -> a, b
This is just a stub that should be defined by an object’s class methods to get anything else.
See Also:

normal return a/b instead of a, b

as_ordered_factors(order=None)
Return list of ordered factors (if Mul) else [self].

as_ordered_terms(order=None, data=False)
Transform an expression to an ordered list of terms.

Examples

>>> from sympy import sin, cos
>>> from sympy.abc import x, y

>>> (sin(x)**2*cos(x) + sin(x)**2 + 1).as_ordered_terms()
[sin(x)**2*cos(x), sin(x)**2, 1]

as_poly(*gens, **args)
Converts self to a polynomial or returns None.
```python
>>> from sympy import Poly, sin
>>> from sympy.abc import x, y

>>> print (x**2 + x*y).as_poly()
Poly(x**2 + x*y, x, y, domain='ZZ')

>>> print (x**2 + x*y).as_poly(x, y)
Pony(x**2 + x*y, x, y, domain='ZZ')

>>> print (x**2 + sin(y)).as_poly(x, y)
None
```

docs.

**as_powers_dict()**
Return self as a dictionary of factors with each factor being treated as a power. The keys are the bases of the factors and the values, the corresponding exponents. The resulting dictionary should be used with caution if the expression is a Mul and contains non-commutative factors since the order that they appeared will be lost in the dictionary.

**as_real_imag**(deep=True, **hints)**

**as_terms()**
Transform an expression to a list of terms.

**assumptions0**

**atoms(**types**)**
Returns the atoms that form the current object.

By default, only objects that are truly atomic and can’t be divided into smaller pieces are returned: symbols, numbers, and number symbols like I and pi. It is possible to request atoms of any type, however, as demonstrated below.

### Examples

```python
>>> from sympy import Number, NumberSymbol, Symbol
>>> (1 + x + 2*sin(y + I*pi)).atoms(Symbol)
set([x, y])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Number)
set([1, 2])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Number, NumberSymbol)
set([1, 2, pi])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Number, NumberSymbol, I)
set([1, 2, I, pi])
```

Note that I (imaginary unit) and zoo (complex infinity) are special types of number symbols and are not part of the NumberSymbol class.

The type can be given implicitly, too:

```python
>>> (1 + x + 2*sin(y + I*pi)).atoms(x)  # x is a Symbol
set([x, y])
```

Be careful to check your assumptions when using the implicit option since `S(1).is_Integer = True` but `type(S(1))` is `One`, a special type of sympy atom, while `type(S(2))` is `type Integer` and will find all integers in an expression:
```python
>>> from sympy import S
>>> (1 + x + 2*sin(y + I*pi)).atoms(S(1))
set([1])

>>> (1 + x + 2*sin(y + I*pi)).atoms(S(2))
set([1, 2])

Finally, arguments to atoms() can select more than atomic atoms: any sympy type (loaded in
core/__init__.py) can be listed as an argument and those types of “atoms” as found in scanning the
arguments of the expression recursively:
```
```python
>>> from sympy import Function, Mul
>>> from sympy.core.function import AppliedUndef

>>> f = Function('f')

>>> (1 + f(x) + 2*sin(y + I*pi)).atoms(Function)
set([f(x), sin(y + I*pi)])

>>> (1 + f(x) + 2*sin(y + I*pi)).atoms(AppliedUndef)
set([f(x)])

>>> (1 + x + 2*sin(y + I*pi)).atoms(Mul)
set([I*pi, 2*sin(y + I*pi)])
```

cancel(*gens, **args)

See the cancel function in sympy.polys
classmethod class_key()

coeff(x, n=1, right=False)

Returns the coefficient from the term containing “x**n” or None if there is no such term. If
n is zero then all terms independent of x will be returned.

When x is noncommutative, the coeff to the left (default) or right of x can be returned. The keyword ‘right’
is ignored when x is commutative.

See Also:

as_coeff_Add a method to separate the additive constant from an expression

as_coeff_Mul a method to separate the multiplicative constant from an expression

as_independent a method to separate x dependent terms/factors from others

Examples

```python
>>> from sympy import symbols
>>> from sympy.abc import x, y, z

You can select terms that have an explicit negative in front of them:
```
```python
>>> (-x + 2*y).coeff(-1)
x

>>> (x - 2*y).coeff(-1)
2*y

You can select terms with no Rational coefficient:
```
```python
>>> (x + 2*y).coeff(1)
x
```
>>> (3 + 2*x + 4*x**2).coeff(1)
0

You can select terms independent of x by making n=0; in this case expr.as_independent(x)[0] is returned (and 0 will be returned instead of None):

>>> (3 + 2*x + 4*x**2).coeff(x, 0)
3
>>> eq = ((x + 1)**3).expand() + 1
>>> eq
x**3 + 3*x**2 + 3*x + 2
>>> [eq.coeff(x, i) for i in reversed(range(4))]
[1, 3, 3, 2]
>>> eq -= 2
>>> [eq.coeff(x, i) for i in reversed(range(4))]
[1, 3, 3, 0]

You can select terms that have a numerical term in front of them:

>>> (-x - 2*y).coeff(2)
y
>>> from sympy import sqrt
>>> (x + sqrt(2)*x).coeff(sqrt(2))
x

The matching is exact:

>>> (3 + 2*x + 4*x**2).coeff(x)
2
>>> (3 + 2*x + 4*x**2).coeff(x**2)
4
>>> (3 + 2*x + 4*x**2).coeff(x**3)
0
>>> (z*(x + y)**2).coeff((x + y)**2)
z
>>> (z*(x + y)**2).coeff(x + y)
0

In addition, no factoring is done, so 1 + z*(1 + y) is not obtained from the following:

>>> (x + z*(x + x*y)).coeff(x)
1

If such factoring is desired, factor_terms can be used first:

>>> from sympy import factor_terms
>>> factor_terms(x + z*(x + x*y)).coeff(x)
z*(y + 1) + 1

>>> n, m, o = symbols(’n m o’, commutative=False)
>>> n.coef(n)
1
>>> (3*n).coeff(n)
3
>>> (n*m + m*n*m).coeff(n) # = (1 + m)*n*m
1 + m
>>> (n*m + m*n*m).coeff(n, right=True) # = (1 + m)*n*m
m

If there is more than one possible coefficient 0 is returned:
```python
>>> (n*m + m*n).coeff(n)
0

If there is only one possible coefficient, it is returned:
```n```> (n*m + x*m*n).coeff(m*n)
x
```n```> (n*m + x*m*n).coeff(m*n, right=1)
1
```

**collect** *(syms, func=None, evaluate=True, exact=False, distribute_order_term=True)*

See the collect function in sympy.simplify

**combsimp** *

See the combsimp function in sympy.simplify

**compare** *(other)*

Return -1, 0, 1 if the object is smaller, equal, or greater than other.

Not in the mathematical sense. If the object is of a different type from the “other” then their classes are ordered according to the sorted_classes list.

**Examples**

```python
>>> from sympy.abc import x, y
```n```> x.compare(y)
-1
```n```> x.compare(x)
0
```n```> y.compare(x)
1
```
```

**static compare.pretty** *(\*args, \*\*kwargs)*

Is a > b in the sense of ordering in printing?

THIS FUNCTION IS DEPRECATED. Use default_sort_key instead.

```python
yes ..... return 1
no ...... return -1
equal ... return 0
```

**Strategy:**

It uses Basic.compare as a fallback, but improves it in many cases, like x**3, x**4, O(x**3) etc. In those simple cases, it just parses the expression and returns the “sane” ordering such as:

```
1 < x < x**2 < x**3 < O(x**4) etc.
```

**Examples**

```python
>>> from sympy.abc import x
```n```> from sympy import Basic, Number
```n```> Basic._compare.pretty(x, x**2)
-1
```n```> Basic._compare.pretty(x**2, x**2)
0
```n```> Basic._compare.pretty(x**3, x**2)
```
>> Basic._compare_pretty(Number(1, 2), Number(1, 3))
1
>> Basic._compare_pretty(Number(0), Number(-1))
1

compute_leading_term(x, skip_abs=False, logx=None)
    as_leading_term is only allowed for results of .series() This is a wrapper to compute a series first. If skip_abs is true, the absolute term is assumed to be zero. (This is necessary because sometimes it cannot be simplified to zero without a lot of work, but is still known to be zero. See log._eval_nseries for an example.) If skip_log is true, log(x) is treated as an independent symbol. (This is needed for the gruntz algorithm.)

conjugate()

copy()

could_extract_minus_sign()
    Canonical way to choose an element in the set {e, -e} where e is any expression. If the canonical element is e, we have e.could_extract_minus_sign() == True, else e.could_extract_minus_sign() == False.
    For any expression, the set {e.could_extract_minus_sign(), (-e).could_extract_minus_sign()} must be (True, False).
    >>> from sympy.abc import x, y
    >>> (x-y).could_extract_minus_sign() != (y-x).could_extract_minus_sign()
    True

count(query)
    Count the number of matching subexpressions.

count_ops(visual=None)
    wrapper for count_ops that returns the operation count.

default_assumptions ={}

diff(*symbols, **assumptions)

doit(**hints)

dummy_eq(other, symbol=None)
    Compare two expressions and handle dummy symbols.

Examples

    >>> from sympy import Dummy
    >>> from sympy.abc import x, y

    >>> u = Dummy('u')

    >>> (u**2 + 1).dummy_eq(x**2 + 1)
    True
    >>> (u**2 + 1) == (x**2 + 1)
    False

    >>> (u**2 + y).dummy_eq(x**2 + y, x)
    True
    >>> (u**2 + y).dummy_eq(x**2 + y, y)
    False

52.2. Classes
equals (other, failing_expression=False)
Return True if self == other, False if it doesn’t, or None. If failing_expression is True then the expression which did not simplify to a 0 will be returned instead of None.

If self is a Number (or complex number) that is not zero, then the result is False.

If self is a number and has not evaluated to zero, evalf will be used to test whether the expression evaluates to zero. If it does so and the result has significance (i.e. the precision is either -1, for a Rational result, or is greater than 1) then the evalf value will be used to return True or False.

evalf (n=15, subs=None, maxn=100, chop=False, strict=False, quad=None, verbose=False)
Evaluate the given formula to an accuracy of n digits. Optional keyword arguments:

sub=<dict> Substitute numerical values for symbols, e.g. subs={x:3, y:1+pi}.

maxn=<integer> Allow a maximum temporary working precision of maxn digits (default=100)

chop=<bool> Replace tiny real or imaginary parts in subresults by exact zeros (default=False)

strict=<bool> Raise PrecisionExhausted if any subresult fails to evaluate to full accuracy, given the available maxprec (default=False)

quad=<str> Choose algorithm for numerical quadrature. By default, tanh-sinh quadrature is used. For oscillatory integrals on an infinite interval, try quad='osc'.

verbose=<bool> Print debug information (default=False)

expand (*args, **kw_args)
Expand an expression using hints.

See the docstring of the expand() function in sympy.core.function for more information.

extract_additively (c)
Return self - c if it’s possible to subtract c from self and make all matching coefficients move towards zero, else return None.

See Also:
extract_multiplicatively, coeff, as_coefficient

Examples

>>> from sympy import S
>>> from sympy.abc import x, y
>>> e = 2*x + 3
>>> e.extract_additively(x + 1)
x + 2
>>> e.extract_additively(3*x)
>>> e.extract_additively(4)
>>> (x + 1).extract_additively(x + 1)
(x + 1)*(x + 2*y) + 3

Sometimes auto-expansion will return a less simplified result than desired; gcd_terms might be used in such cases:

>>> from sympy import gcd_terms
>>> (4*x*(y + 1) + y).extract_additively(x)
4*x*(y + 1) + x*(4*y + 3) - x*(4*y + 4) + y
>>> gcd_terms(_)
x*(4*y + 3) + y
extract_branch_factor (allow_half=False)
  Try to write self as \exp(2\pi\text{i}n)z in a nice way. Return (z, n).

  >>> from sympy import exp_polar, I, pi
  >>> from sympy.abc import x, y
  >>> exp_polar(I*pi).extract_branch_factor()
  (exp_polar(I*pi), 0)
  >>> exp_polar(2*I*pi).extract_branch_factor()
  (1, 1)
  >>> exp_polar(-I*pi).extract_branch_factor()
  (exp_polar(I*pi), -1)
  >>> exp_polar(3*I*pi + x).extract_branch_factor()
  (exp_polar(x + I*pi), 1)
  >>> (y*exp_polar(-5*pi*I)*exp_polar(3*pi*I + 2*pi*x)).extract_branch_factor()
  (y*exp_polar(2*pi*x), -1)
  >>> exp_polar(-I*pi/2).extract_branch_factor()
  (exp_polar(-I*pi/2), 0)

  If allow_half is True, also extract \exp(I\pi):

  >>> exp_polar(I*pi).extract_branch_factor(allow_half=True)
  (1, 1/2)
  >>> exp_polar(2*I*pi).extract_branch_factor(allow_half=True)
  (1, 1)
  >>> exp_polar(3*I*pi).extract_branch_factor(allow_half=True)
  (1, 3/2)
  >>> exp_polar(-I*pi).extract_branch_factor(allow_half=True)
  (1, -1/2)

extract_multiplicatively (c)
  Return None if it’s not possible to make self in the form c * something in a nice way, i.e. preserving the properties of arguments of self.

  >>> from sympy import symbols, Rational
  >>> x, y = symbols(’x,y’, real=True)
  >>> (x*y)**3.extract_multiplicatively(x**2 * y)
  x*y**2
  >>> (x*y)**3.extract_multiplicatively(x**4 * y)
  >>> (2*x).extract_multiplicatively(2)
  x
  >>> (2*x).extract_multiplicatively(3)
  >>> (Rational(1,2)*x).extract_multiplicatively(3)
  x/6

factor (*gens, **args)
  See the factor() function in sympy.polys.polytools

find (query, group=False)
  Find all subexpressions matching a query.

formula
  Return a Formula with only terms=[self].

free_symbols
**classmethod** `fromiter` *(args, **assumptions)*

Create a new object from an iterable.

This is a convenience function that allows one to create objects from any iterable, without having to convert to a list or tuple first.

**Examples**

```python
>>> from sympy import Tuple
>>> Tuple.fromiter(i for i in xrange(5))
(0, 1, 2, 3, 4)
```

**func**

The top-level function in an expression.

The following should hold for all objects:

```python
>> x == x.func(*x.args)
```

**Examples**

```python
>>> from sympy.abc import x
>>> a = 2*x
>>> a.func
<class 'sympy.core.mul.Mul'>
>>> a.args
(2, x)
>>> a.func(*a.args)
2*x
>>> a == a.func(*a.args)
True
```

**getO()**

Returns the additive O(...) symbol if there is one, else None.

**getn()**

Returns the order of the expression.

The order is determined either from the O(...) term. If there is no O(...) term, it returns None.

**Examples**

```python
>>> from sympy import O
>>> from sympy.abc import x
>>> (1 + x + O(x**2)).getn()
2
>>> (1 + x).getn()
```

**has(** *args, **kw_args**)

Test whether any subexpression matches any of the patterns.
Examples

```python
>>> from sympy import sin, S
>>> from sympy.abc import x, y, z
>>> (x**2 + sin(x*y)).has(z)
False
>>> (x**2 + sin(x*y)).has(x, y, z)
True
>>> x.has(x)
True
```

Note that `expr.has(*patterns)` is exactly equivalent to `any(expr.has(p) for p in patterns)`. In particular, `False` is returned when the list of patterns is empty.

```python
>>> x.has()
False
```

`integrate(*args, **kwargs)`
See the integrate function in sympy.integrals

`invert(g)`
See the invert function in sympy.polys

`is_Add = False`
`is_AlgebraicNumber = False`
`is_Atomic = True`
`is_Boolean = False`
`is_Derivative = False`
`is_Dummy = False`
`is_Equality = False`
`is_Float = False`
`is_Function = False`
`is_Integer = False`
`is_Matrix = False`
`is_Mul = False`
`is_Not = False`
`is_Number = False`
`is_NumberSymbol = False`
`is_Order = False`
`is_Piecewise = False`
`is_Poly = False`
`is_Pow = False`
`is_Rational = False`
`is_Real`
    Deprecated alias for `is_Float`
`is_Relational = False`
is_Symbol = True
is_Wild = False
is_antihermitian
is_bounded
is_commutative
is_comparable = False
is_complex
is_composite
is_constant (*wrt, *flags)
is_even
is_finite
is_hermitian
is_hypergeometric (k)
is_imaginary
is_infinitesimal
is_integer
is_irrational
is_negative
is_noninteger
is_nonnegative
is_nonpositive
is_nonzero
is_number
is_odd
is_polar
is_polynomial (*sym)
    Return True if self is a polynomial in sym and False otherwise.

This checks if self is an exact polynomial in sym. This function returns False for expressions that
are “polynomials” with symbolic exponents. Thus, you should be able to apply polynomial algorithms
to expressions for which this returns True, and Poly(expr, *sym) should work only if and only if
expr.is_polynomial(*sym) returns True. The polynomial does not have to be in expanded form. If no
symbols are given, all free symbols in the expression will be used.

This is not part of the assumptions system. You cannot do Symbol(‘z’, polynomial=True).

Examples
>>> from sympy import Symbol
>>> x = Symbol('x')
>>> ((x**2 + 1)**4).is_polynomial(x)
True
>>> ((x**2 + 1)**4).is_polynomial()
True
>>> (2**x + 1).is_polynomial(x)
False

>>> n = Symbol('n', nonnegative=True, integer=True)
>>> (x**n + 1).is_polynomial(x)
False

This function does not attempt any nontrivial simplifications that may result in an expression that does not
appear to be a polynomial to become one.

>>> from sympy import sqrt, factor, cancel
>>> y = Symbol('y', positive=True)
>>> a = sqrt(y**2 + 2*y + 1)
>>> a.is_polynomial(y)
False
>>> factor(a)
y + 1
>>> factor(a).is_polynomial(y)
True

>>> b = (y**2 + 2*y + 1)/(y + 1)
>>> b.is_polynomial(y)
False
>>> cancel(b)
y + 1
>>> cancel(b).is_polynomial(y)
True

See also .is_rational_function()

is_positive

is_prime

is_rational

is_rational_function(*syms)

Test whether function is a ratio of two polynomials in the given symbols, syms. When syms is not given,
all free symbols will be used. The rational function does not have to be in expanded or in any kind of
canonical form.

This function returns False for expressions that are “rational functions” with symbolic exponents. Thus,
you should be able to call .as_numer_denom() and apply polynomial algorithms to the result for expres-
sions for which this returns True.

This is not part of the assumptions system. You cannot do Symbol(‘z’, rational_function=True).

Examples

>>> from sympy import Symbol, sin
>>> from sympy.abc import x, y
Neuroimaging in Python Documentation, Release 0.3.0

```python
>>> (x/y).is_rational_function()
True

>>> (x**2).is_rational_function()
True

>>> (x/sin(y)).is_rational_function(y)
False

>>> n = Symbol('n', integer=True)
>>> (x**n + 1).is_rational_function(x)
False
```

This function does not attempt any nontrivial simplifications that may result in an expression that does not appear to be a rational function to become one.

```python
>>> from sympy import sqrt, factor, cancel

>>> y = Symbol('y', positive=True)
>>> a = sqrt(y**2 + 2*y + 1)/y
>>> a.is_rational_function(y)
False

>>> factor(a)
(y + 1)/y

>>> factor(a).is_rational_function(y)
True
```

See also `is_rational_function()`.

**is_real**

**is_unbounded**

**is_zero**

**iter_basic_args()**

Iterates arguments of `self`.

**Examples**

```python
>>> from sympy.abc import x

>>> a = 2*x

>>> a.iter_basic_args()
<...iterator object at 0x...>

>>> list(a.iter_basic_args())
[2, x]
```

**leadterm(x)**

Returns the leading term `a*x**b` as a tuple `(a, b)`.

**Examples**

```python
>>> from sympy.abc import x

>>> (1+x+x**2).leadterm(x)
(1, 0)

>>> (1/x**2+x+x**2).leadterm(x)
(1, -2)
```
**limit** \((x, \text{xlim}, \text{dir}=‘+’)\)
Compute limit \(x\to\text{xlim}\).

**lseries** \((x=\text{None}, x0=0, \text{dir}=‘+’)\)
Wrapper for series yielding an iterator of the terms of the series.
Note: an infinite series will yield an infinite iterator. The following, for example, will never terminate. It will just keep printing terms of the \(\sin(x)\) series:

```python
for term in \sin(x).lseries(x):
    print term
```

The advantage of `lseries()` over `nseries()` is that many times you are just interested in the next term in the series (i.e. the first term for example), but you don’t know how many you should ask for in `nseries()` using the “\(n\)” parameter.
See also `nseries()`.

**match** \((\text{pattern})\)
Pattern matching.
Wild symbols match all.
Return `None` when expression \((\text{self})\) does not match with pattern. Otherwise return a dictionary such that:

```python
\text{pattern.xreplace(\text{self.match(\text{pattern})) == self}
```

**Examples**

```python
>>> from sympy import symbols, Wild
>>> from sympy import symbols, Wild
>>> from sympy.abc import x, y
>>> p = Wild("p")
>>> q = Wild("q")
>>> r = Wild("r")
>>> e = (x+y)**(x+y)
>>> e.match(p**p)
{p_: x + y}
>>> e.match(p**q)
{p_: x + y, q_: x + y}
>>> e = (2*x)**2
>>> e.match(p*q**r)
{p_: 4, q_: x, r_: 2}
>>> (p*q**r).xreplace(e.match(p*q**r))
4*x**2
```

**matches** \((\text{expr, repl_dict={}})\)

Evaluate the given formula to an accuracy of \(n\) digits. Optional keyword arguments:

- `subs=dict` ➔ Substitute numerical values for symbols, e.g. \(\text{subs=\{x:3, y:1+pi\}}\).
- `maxn=<integer>` ➔ Allow a maximum temporary working precision of \(\text{maxn}\) digits (default=100)
- `chop=<bool>` ➔ Replace tiny real or imaginary parts in subresults by exact zeros (default=False)
- `strict=<bool>` ➔ Raise `PrecisionExhausted` if any subresult fails to evaluate to full accuracy, given the available `maxprec` (default=False)
- `quad=<str>` ➔ Choose algorithm for numerical quadrature. By default, tanh-sinh quadrature is used. For oscillatory integrals on an infinite interval, try `quad=‘osc’`.

---

52.2. Classes 411
verbose=<bool>  Print debug information (default=False)

name

nseries (x=None, x0=0, n=6, dir='+', logx=None)
Wrapper to _eval_nseries if assumptions allow, else to series.
If x is given, x0 is 0, dir='+', and self has x, then _eval_nseries is called. This calculates “n” terms in the
innermost expressions and then builds up the final series just by “cross-multiplying” everything out.
Advantage – it’s fast, because we don’t have to determine how many terms we need to calculate in advance.
Disadvantage – you may end up with less terms than you may have expected, but the O(x**n) term appended will always be correct and so the result, though perhaps shorter, will also be correct.
If any of those assumptions is not met, this is treated like a wrapper to series which will try harder to return
the correct number of terms.
See also lseries().

nsimplify (constants=[], tolerance=None, full=False)
See the nsimplify function in sympy.simplify

powsimp (deep=False, combine='all')
See the powsimp function in sympy.simplify

primitive ()
Return the positive Rational that can be extracted non-recursively from every term of self (i.e., self is
treated like an Add). This is like the as_coeff_Mul() method but primitive always extracts a positive
Rational (never a negative or a Float).

Examples

>>> from sympy.abc import x
>>> (3*(x + 1)**2).primitive()
(3, (x + 1)**2)
>>> a = (6*x + 2); a.primitive()
(2, 3*x + 1)
>>> b = (x/2 + 3); b.primitive()
(1/2, x + 6)
>>> (a*b).primitive() == (1, a*b)
True

radsimp ()
See the radsimp function in sympy.simplify

ratsimp ()
See the ratsimp function in sympy.simplify

refine (assumption=True)
See the refine function in sympy.assumptions

removeO ()
Removes the additive O(..) symbol if there is one

replace (query, value, map=False)
Replace matching subexpressions of self with value.
If map = True then also return the mapping {old: new} where old was a sub-expression found with
query and new is the replacement value for it.
Traverses an expression tree and performs replacement of matching subexpressions from the bottom to the
top of the tree. The list of possible combinations of queries and replacement values is listed below:

See Also:

sub  substitution of subexpressions as defined by the objects themselves.
xreplace  exact node replacement in expr tree; also capable of using matching rules

Examples

Initial setup

```python
>>> from sympy import log, sin, cos, tan, Wild
>>> from sympy.abc import x, y

f = log(sin(x)) + tan(sin(x**2))
```

1.1. type -> type  obj.replace(sin, tan)

```python
>>> f.replace(sin, cos)
log(cos(x)) + tan(cos(x**2))
```

1.2. type -> func  obj.replace(sin, lambda arg: ...)

```python
>>> f.replace(sin, lambda arg: sin(2*arg))
log(sin(2*x)) + tan(sin(2*x**2))
```

2.1. expr -> expr  obj.replace(sin(a), tan(a))

```python
>>> a = Wild('a')
>>> f.replace(sin(a), tan(a))
log(tan(x)) + tan(tan(x**2))
```

2.2. expr -> func  obj.replace(sin(a), lambda a: ...)

```python
>>> f.replace(sin(a), cos(a))
log(cos(x)) + tan(cos(x**2))
```

3.1. func -> func  obj.replace(lambda expr: ..., lambda expr: ...)

```python
>>> g = 2*sin(x**3)
>>> g.replace(lambda expr: expr.is_Number, lambda expr: expr**2)
4*sin(x**9)
```

rewrite (*args, **hints)

Rewrites expression containing applications of functions of one kind in terms of functions of different kind.
For example you can rewrite trigonometric functions as complex exponentials or combinatorial functions
as gamma function.

As a pattern this function accepts a list of functions to to rewrite (instances of DefinedFunction class). As
rule you can use string or a destination function instance (in this case rewrite() will use the str() function).
There is also possibility to pass hints on how to rewrite the given expressions. For now there is only one such hint defined called ‘deep’. When ‘deep’ is set to False it will forbid functions to rewrite their contents.

```python
>>> from sympy import sin, exp, I
>>> from sympy.abc import x, y

>>> sin(x).rewrite(sin, exp)
-I*(exp(I*x) - exp(-I*x))/2
```

**round** *(p=0)*

Return x rounded to the given decimal place.

If a complex number would results, apply round to the real and imaginary components of the number.

**Notes**

Do not confuse the Python builtin function, round, with the SymPy method of the same name. The former always returns a float (or raises an error if applied to a complex value) while the latter returns either a Number or a complex number:

```python
>>> isinstance(round(S(123), -2), Number)
False
>>> isinstance(S(123).round(-2), Number)
True
>>> isinstance((3*I).round(), Mul)
True
>>> isinstance((1 + 3*I).round(), Add)
True
```

**Examples**

```python
>>> from sympy import pi, E, I, S, Add, Mul, Number

>>> S(10.5).round()
11.
>>> pi.round()
3.
>>> pi.round(2)
3.14
>>> (2*pi + E*I).round()
6. + 3.*I
```

The round method has a chopping effect:

```python
>>> (2*pi + I/10).round()
6.
>>> (pi/10 + 2*I).round()
2.*I
>>> (pi/10 + E*I).round(2)
0.31 + 2.72*I
```

**separate** *(deep=False, force=False)*

See the separate function in sympy.simplify

**series** *(x=None, x0=0, n=6, dir='+')*

Series expansion of “self” around x = x0 yielding either terms of the series one by one (the lazy series given when n=None), else all the terms at once when n != None.
Note: when \( n \neq None \), if an \( O() \) term is returned then the \( x \) in the \( \text{in} \) it and the entire expression represents \( x - x_0 \), the displacement from \( x_0 \). (If there is no \( O() \) term then the series was exact and \( x \) has it’s normal meaning.) This is currently necessary since sympy’s \( O() \) can only represent terms at \( x_0=0 \). So instead of:

\[
\cos(x).\text{series}(x0=1, n=2) \rightarrow (1 - x)\sin(1) + \cos(1) + O((x - 1)^2)
\]

which graphically looks like this:

```
|   |
. . |   
. . |   |
----+--------------
| . . . . .    |
| x=0          |
```

the following is returned instead:

\[-x\sin(1) + \cos(1) + O(x^2)\]

whose graph is this:

```
\ \ |   |
. . |   
. . |   |
-----\--------------
| . . . . .    |
| x=0          |
```

which is identical to \( \cos(x + 1)\).\text{series}(n=2) \).

**Usage:** Returns the series expansion of “self” around the point \( x = x_0 \) with respect to \( x \) up to \( O(x^n) \) (default \( n \) is 6).

If \( x=None \) and self is univariate, the univariate symbol will be supplied, otherwise an error will be raised.

```python
>>> from sympy import cos, exp
>>> from sympy.abc import x, y
>>> cos(x).series() 1 - x**2/2 + x**4/24 + O(x**6)
>>> cos(x).series(n=4) 1 - x**2/2 + O(x**4)
>>> e = cos(x + exp(y))
>>> e.series(y, n=2) cos(x + 1) - y*sin(x + 1) + O(y**2)
>>> e.series(x, n=2) cos(exp(y)) - x*sin(exp(y)) + O(x**2)
```

If \( n=None \) then an iterator of the series terms will be returned.

```python
>>> term=cos(x).series(n=None)
>>> [term.next() for i in range(2)]
[1, -x**2/2]
```

For \( \text{dir=}+ \) (default) the series is calculated from the right and for \( \text{dir=}-- \) the series from the left. For smooth functions this flag will not alter the results.
```python
>>> abs(x).series(dir="+")
-x
>>> abs(x).series(dir="-")
-x
```

`simplify()`
See the `simplify` function in sympy.simplify

`sort_key` (*args, **kw_args)

`simplify()`
See the `simplify` function in sympy.simplify

`simplify()`
See the `simplify` function in sympy.simplify

`subs` (*args, **kwargs)
Substitutes old for new in an expression after sympifying args.

`args` is either:

- two arguments, e.g. foo.subs(old, new)
- one iterable argument, e.g. foo.subs(iterable). The iterable may be
  - an iterable container with (old, new) pairs. In this case the replacements are processed
    in the order given with successive patterns possibly affecting replacements already made.
  - a dict or set whose key/value items correspond to old/new pairs. In this case the old/new
    pairs will be sorted by op count and in case of a tie, by number of args and the default_sort_key.
    The resulting sorted list is then processed as an iterable container (see previous).

If the keyword `simultaneous` is True, the subexpressions will not be evaluated until all the substitutions
have been made.

See Also:

- `replace` replacement capable of doing wildcard-like matching, parsing of match, and conditional re-
  placements
- `xreplace` exact node replacement in expr tree; also capable of using matching rules

**Examples**

```python
>>> from sympy import pi, exp
>>> from sympy.abc import x, y

>>> (1 + x*y).subs(x, pi)
pi*y + 1
>>> (1 + x*y).subs({x:pi, y:2})
1 + 2*pi
>>> (1 + x*y).subs([(x, pi), (y, 2)])
1 + 2*pi
>>> reps = [(y, x**2), (x, 2)]
>>> (x + y).subs(reps)
6
>>> (x + y).subs(reversed(reps))
x**2 + 2
>>> (x**2 + x**4).subs(x**2, y)
y**2 + y
```

To replace only the `x**2` but not the `x**4`, use `xreplace`:

```python
>>> (x**2 + x**4).xreplace((x**2: y))
x**4 + y
```
To delay evaluation until all substitutions have been made, set the keyword `simultaneous` to True:

```python
>>> (x/y).subs([(x, 0), (y, 0)])
0
>>> (x/y).subs([(x, 0), (y, 0)], simultaneous=True)
nan
```

This has the added feature of not allowing subsequent substitutions to affect those already made:

```python
>>> ((x + y)/y).subs({x + y: y, y: x + y})
1
>>> ((x + y)/y).subs({x + y: y, y: x + y}, simultaneous=True)
y/(x + y)
```

In order to obtain a canonical result, unordered iterables are sorted by `count_op` length, number of arguments and by the `default_sort_key` to break any ties. All other iterables are left unsorted.

```python
>>> from sympy import sqrt, sin, cos, exp
>>> from sympy.abc import a, b, c, d, e

>>> A = (sqrt(sin(2*x)), a)
>>> B = (sin(2*x), b)
>>> C = (cos(2*x), c)
>>> D = (x, d)
>>> E = (exp(x), e)

>>> expr = sqrt(sin(2*x))*sin(exp(x)*x)*cos(2*x) + sin(2*x)

>>> expr.subs(dict([A,B,C,D,E]))
a*c*sin(d*e) + b
```

### together(*args, **kwargs)

See the `together` function in `sympy.polys`

### transpose()

### trigsimp(deep=False, recursive=False)

See the `trigsimp` function in `sympy.simplify`

### xreplace(rule)

### 52.3 Functions

*nipy.algorithms.statistics.formula.formulae.contrast_from_cols_or_rows(L, D, pseudo=None)*

Construct a contrast matrix from a design matrix D

(possibly with its pseudo inverse already computed) and a matrix L that either specifies something in the column space of D or the row space of D.

**Parameters**

- **L** : ndarray
  - Matrix used to try and construct a contrast.

- **D** : ndarray
  - Design matrix used to create the contrast.

- **pseudo** : None or array-like, optional
If not None, gives pseudo-inverse of \( D \). Allows you to pass this if it is already calculated.

**Returns**  
\( C : \text{ndarray} \)  

**Notes**

From an \( n \times p \) design matrix \( D \) and a matrix \( L \), tries to determine a \( p \times q \) contrast matrix \( C \) which determines a contrast of full rank, i.e. the \( n \times q \) matrix

\[
\text{dot} (\text{transpose}(C), \text{pinv}(D))
\]

is full rank.

\( L \) must satisfy either \( L.shape[0] == n \) or \( L.shape[1] == p \).

If \( L.shape[0] == n \), then \( L \) is thought of as representing columns in the column space of \( D \).

If \( L.shape[1] == p \), then \( L \) is thought of as what is known as a contrast matrix. In this case, this function returns an estimable contrast corresponding to the \( \text{dot}(D, L.T) \)

This always produces a meaningful contrast, not always with the intended properties because \( q \) is always non-zero unless \( L \) is identically 0. That is, it produces a contrast that spans the column space of \( L \) (after projection onto the column space of \( D \)).

**Examples**

```python
>>> x, y, z = [Term(l) for l in 'xyz']
>>> f = Formula([x,y,z])
>>> getparams(f)
[]
>>> f.mean
_b0*x + _b1*y + _b2*z
>>> getparams(f.mean)
[_b0, _b1, _b2]
>>> th = sympy.Symbol('theta')
>>> f.mean*sympy.exp(th)
(_b0*x + _b1*y + _b2*z)*exp(theta)
>>> getparams(f.mean*sympy.exp(th))
[theta, _b0, _b1, _b2]
```

**Examples**

```python
>>> x, y, z = [Term(l) for l in 'xyz']
>>> f = Formula([x,y,z])
>>> getterms(f)
[x, y, z]
>>> getterms(f.mean)
[x, y, z]
```
nipy.algorithms.statistics.formula.formulae.is_factor(obj)
Is obj a Factor?

nipy.algorithms.statistics.formula.formulae.is_factor_term(obj)
Is obj a FactorTerm?

nipy.algorithms.statistics.formula.formulae.is_formula(obj)
Is obj a Formula?

nipy.algorithms.statistics.formula.formulae.is_term(obj)
Is obj a Term?

nipy.algorithms.statistics.formula.formulae.make_dummy(name)
Make dummy variable of given name

Parameters name : str
name of dummy variable

Returns dum : Dummy instance

Notes
The interface to Dummy changed between 0.6.7 and 0.7.0

nipy.algorithms.statistics.formula.formulae.make_recarray(rows, names, dtyes=None)
Create recarray from rows with field names names

Create a recarray with named columns from a list of rows and names for the columns. If dtype is None, the
dtype is based on rows if it is an np.ndarray, else the data is cast as np.float. If dtypes are supplied, it uses the
dtypes to create a np.dtype unless rows is an np.ndarray, in which case dtypes are ignored

Parameters rows: array-like :
Rows that will be turned into an recarray.

names: sequence :
Sequence of strings - names for the columns.

dtypes: [str or np.dtype] :
Used to create a np.dtype, can be np.dtypes or string.

Returns v : np.ndarray

Examples
The following tests depend on machine byte order to pass

>>> arr = np.array([[3, 4], [4, 6], [6, 8]])
>>> make_recarray(arr, ['x', 'y'])
array([[3, 4], [4, 6], [6, 8]], dtype=[('x', '<i8'), ('y', '<i8')])
>>> r = make_recarray(arr, ['w', 'u'])
>>> make_recarray(r, ['x', 'y'])
array([[3, 4],
       [4, 6],
       [6, 8]],
      dtype=('x', '...'), ('y', '...'))

>>> make_recarray([[3, 4], [4, 6], [7, 9]], 'wv', [np.float, np.int])
array([[3.0, 4], [4.0, 6], [7.0, 9]],
      dtype=('w', '...'), ('v', '...'))

nipy.algorithms.statistics.formula.formulae.natural_spline(t, knots=None, order=3, intercept=False)

Return a Formula containing a natural spline
Spline for a Term with specified knots and order.

Parameters

- **t**: Term
  - knots : None or sequence, optional
    Sequence of float. Default None (same as empty list)
  - order : int, optional
    Order of the spline. Defaults to a cubic (=3)
  - intercept : bool, optional
    If True, include a constant function in the natural spline. Default is False

Returns

- **formula**: Formula
  A Formula with (len(knots) + order) Terms (if intercept=False, otherwise includes one more Term), made up of the natural spline functions.

Examples

```python
>>> x = Term('x')
>>> n = natural_spline(x, knots=[1,3,4], order=3)
>>> xval = np.array([3,5,7]).view(np.dtype([('x', np.float)]))
>>> n.design(xval, return_float=True)
array([[ 3., 9., 27., 8., 0., -0.],
       [ 5., 25., 125., 64., 8., 1.],
       [ 7., 49., 343., 216., 64., 27.]]

>>> d = n.design(xval)
>>> print d.dtype.descr
[('ns_1(x)', '<f8'), ('ns_2(x)', '<f8'), ('ns_3(x)', '<f8'), ('ns_4(x)', '<f8'), ('ns_5(x)', '<f8')]
```

nipy.algorithms.statistics.formula.formulae.terms(names, **kwargs)

Return list of terms with names given by names

This is just a convenience in defining a set of terms, and is the equivalent of sympy.symbols for defining symbols in sympy.

We enforce the sympy 0.7.0 behavior of returning symbol “abc” from input “abc”, rather than 3 symbols “a”, “b”, “c”.

Parameters

- **names**: str or sequence of str
  - If a single str, can specify multiple “Term”’s with string containing space or ‘,’ as separator.

- **kwargs**: keyword arguments
keyword arguments as for sympy.symbols

Returns ts : Term or tuple

Term instance or list of Term instance objects named from names

Examples

```python
>>> terms(('a', 'b', 'c'))
(a, b, c)
>>> terms('a, b, c')
(a, b, c)
>>> terms('abc')
abc
```
53.1 Module: `algorithms.statistics.mixed_effects_stat`

Inheritance diagram for `nipy.algorithms.statistics.mixed_effects_stat`:

```python
Module for computation of mixed effects statistics with an EM algorithm. i.e. solves problems of the form y = X beta + e1 + e2, where X and Y are known, e1 and e2 are centered with diagonal covariance. V1 = var(e1) is known, and V2 = var(e2) = lambda identity. the code estimates beta and lambda using an EM algorithm. Likelihood ratio tests can then be used to test the columns of beta.

Author: Bertrand Thirion, 2012.

```python
N, P = 15, 500  
V1 = np.random.randn(N, P) ** 2  
effects = np.random.randn(P) > 0  
Y = generate_data(np.ones(N), effects, .25, V1)  
T1 = one_sample_ttest(Y, V1, n_iter=5)  
T2 = [t_stat(Y)[effects == x] for x in np.unique(effects)]  
assert np.array([t1.std() < t2.std() for t1, t2 in zip(T1, T2)]).all()
```

53.2 Class

53.3 MixedEffectsModel

```python
class nipy.algorithms.statistics.mixed_effects_stat.MixedEffectsModel(X, n_iter=5, verbose=False)
```

Bases: object
Class to handle multiple one-sample mixed effects models

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fit(Y, V1)</td>
<td>Launches the EM algorithm to estimate self</td>
</tr>
<tr>
<td>log_like(Y, V1)</td>
<td>Compute the log-likelihood of (Y, V1) under the model</td>
</tr>
<tr>
<td>predict(Y, V1)</td>
<td>Return the log-likelihood of the data. See the log_like method</td>
</tr>
<tr>
<td>score(Y, V1)</td>
<td>Return the log-likelihood of the data.</td>
</tr>
</tbody>
</table>

__init__ (X, n_iter=5, verbose=False)
Set the effects and first-level variance, and initialize related quantities

Parameters

- X: array of shape(n_samples, n_effects),
  the design matrix
- n_iter: int, optional,
  number of iterations of the EM algorithm
- verbose: bool, optional, verbosity mode

fit(Y, V1)
Launches the EM algorithm to estimate self

Parameters

- Y, array of shape (n_samples, n_tests) or (n_samples),
  the estimated effects
- V1, array of shape (n_samples, n_tests) or (n_samples),
  first-level variance

Returns

- self:

log_like(Y, V1)
Compute the log-likelihood of (Y, V1) under the model

Parameters

- Y, array of shape (n_samples, n_tests) or (n_samples),
  the estimated effects
- V1, array of shape (n_samples, n_tests) or (n_samples),
  first-level variance

Returns

- logl: array of shape n_tests,
  the log-likelihood of the model

predict(Y, V1)
Return the log-likelihood of the data. See the log_like method

score(Y, V1)
Return the log-likelihood of the data. See the log_like method

53.4 Functions

nipy.algorithms.statistics.mixed_effects_stat.check_arrays(Y, V1)
Check that the given data can be used for the models
Parameters  

- **Y**: array of shape (n_samples, n_tests) or (n_samples):
  - the estimated effects

- **V1**: array of shape (n_samples, n_tests) or (n_samples):
  - first-level variance

```python
nipy.algorithms.statistics.mixed_effects_stat.generate_data(X, beta, V2, V1)
```

Generate a group of individuals from the provided parameters

Parameters  

- **X**: array of shape (n_samples, n_reg),
  - the design matrix of the model

- **beta**: float or array of shape (n_reg, n_tests),
  - the associated effects

- **V2**: float or array of shape (n_tests),
  - group variance

- **V1**: array of shape(n_samples, n_tests),
  - the individual variances

Returns  

- **Y**: array of shape(n_samples, n_tests):
  - the individual data related to the two-level normal model

```python
nipy.algorithms.statistics.mixed_effects_stat.mfx_stat(Y, V1, X, column, n_iter=5, return_t=True, return_f=False, return_effect=False, return_var=False, verbose=False)
```

Run a mixed-effects model test on the column of the design matrix

Parameters  

- **Y**: array of shape (n_samples, n_tests):
  - the data

- **V1**: array of shape (n_samples, n_tests):
  - first-level variance associated with the data

- **X**: array of shape(n_samples, n_regressors):
  - the design matrix of the model

- **column**: int,
  - index of the column of X to be tested

- **n_iter**: int, optional,
  - number of iterations of the EM algorithm

- **return_t**: bool, optional,
  - should one return the t test (True by default)

- **return_f**: bool, optional,
  - should one return the F test (False by default)

- **return_effect**: bool, optional,
  - should one return the effect estimate (False by default)
return_var: bool, optional, :
    should one return the variance estimate (False by default)

verbose: bool, optional, verbosity mode :

Returns  (tstat, fstat, effect, var): tuple of arrays of shape (n_tests), :
    those required by the input return booleans
nipy.algorithms.statistics.mixed_effects_stat.one_sample_ftest (Y, V1, n_iter=5, verbose=False)
    Returns the mixed effects F-stat for each row of the X (one sample test) This uses the Formula in Roche et al., NeuroImage 2007

Parameters  Y: array of shape (n_samples, n_tests) :
    the data

V1: array of shape (n_samples, n_tests) :
    first-level variance associated with the data

n_iter: int, optional, :
    number of iterations of the EM algorithm

verbose: bool, optional, verbosity mode :

Returns  fstat, array of shape (n_tests), :
    statistical values obtained from the likelihood ratio test

sign, array of shape (n_tests), :
    sign of the mean for each test (allow for post-hoc signed tests)

nipy.algorithms.statistics.mixed_effects_stat.one_sample_ttest (Y, V1, n_iter=5, verbose=False)
    Returns the mixed effects t-stat for each row of the X (one sample test) This uses the Formula in Roche et al., NeuroImage 2007

Parameters  Y: array of shape (n_samples, n_tests) :
    the observations

V1: array of shape (n_samples, n_tests) :
    first-level variance associated with the observations

n_iter: int, optional, :
    number of iterations of the EM algorithm

verbose: bool, optional, verbosity mode :

Returns  tstat: array of shape (n_tests), :
    statistical values obtained from the likelihood ratio test

nipy.algorithms.statistics.mixed_effects_stat.t_stat (Y)
    Returns the t stat of the sample on each row of the matrix

Parameters  Y, array of shape (n_samples, n_tests) :

Returns  t_variates, array of shape (n_tests) :
nipy.algorithms.statistics.mixed_effects_stat.two_sample_ftest(Y, V1, group, n_iter=5, verbose=False)

Returns the mixed effects t-stat for each row of the X (one sample test) This uses the Formula in Roche et al., NeuroImage 2007

Parameters  

Y: array of shape (n_samples, n_tests) :
the data

V1: array of shape (n_samples, n_tests) :
first-level variance associated with the data

group: array of shape (n_samples) :
a vector of indicators yielding the samples membership

n_iter: int, optional, :
number of iterations of the EM algorithm

verbose: bool, optional, verbosity mode :

Returns  
tstat: array of shape (n_tests), :
statistical values obtained from the likelihood ratio test

nipy.algorithms.statistics.mixed_effects_stat.two_sample_ttest(Y, V1, group, n_iter=5, verbose=False)

Returns the mixed effects t-stat for each row of the X (one sample test) This uses the Formula in Roche et al., NeuroImage 2007

Parameters  

Y: array of shape (n_samples, n_tests) :
the data

V1: array of shape (n_samples, n_tests) :
first-level variance associated with the data

group: array of shape (n_samples) :
a vector of indicators yielding the samples membership

n_iter: int, optional, :
number of iterations of the EM algorithm

verbose: bool, optional, verbosity mode :

Returns  
tstat: array of shape (n_tests), :
statistical values obtained from the likelihood ratio test
54.1 Module: `nipy.algorithms.statistics.models.family.family`

Inheritance diagram for `nipy.algorithms.statistics.models.family.family`:

![Inheritance Diagram]

54.2 Classes

54.2.1 Binomial

```python
class Binomial
    nipy.algorithms.statistics.models.family.family.Binomial(link=<nipy.algorithms.statistics.models.family.links.Logit object at 0x5217e90>, n=1)
```

**Bases:** `nipy.algorithms.statistics.models.family.family.Family`

Binomial exponential family.

**INPUTS:**
- `link` – a Link instance
- `n` – number of trials for Binomial
Methods

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviance</td>
<td>Deviance of (Y, mu) pair. Deviance is usually defined</td>
</tr>
<tr>
<td>devresid</td>
<td>Binomial deviance residual</td>
</tr>
<tr>
<td>fitted</td>
<td>Fitted values based on linear predictors eta.</td>
</tr>
<tr>
<td>predict</td>
<td>Linear predictors based on given mu values.</td>
</tr>
<tr>
<td>variance</td>
<td>Binomial variance function</td>
</tr>
<tr>
<td>weights</td>
<td>Weights for IRLS step.</td>
</tr>
</tbody>
</table>

```python
__init__ (link=<nipy.algorithms.statistics.models.family.links.Logit object at 0x5217e90>, n=1)
```

**deviance (Y, mu, scale=1.0)**

Deviance of (Y, mu) pair. Deviance is usually defined as the difference

\[ DEV = \frac{\text{SUM_i} -2 \log \text{Likelihood}(Y_i,mu_i) + 2 \log \text{Likelihood}(mu_i,mu_i))}{\text{scale}} \]

**INPUTS:**
- Y – response variable
- mu – mean parameter
- scale – optional scale in denominator of deviance

**OUTPUTS:**
- dev

**devresid (Y, mu)**

Binomial deviance residual

**INPUTS:**
- Y – response variable
- mu – mean parameter

**OUTPUTS:**
- resid

**fitted (eta)**

Fitted values based on linear predictors eta.

**INPUTS:**
- eta – values of linear predictors, say, X beta in a generalized linear model.

**OUTPUTS:**
- mu – link.inverse(eta), mean parameter based on eta

**link**

**links** = [Logit, CLogLog]

**predict (mu)**

Linear predictors based on given mu values.

**INPUTS:**
- mu – mean parameter of one-parameter exponential family

**OUTPUTS:**
- eta

**tol** = 1e-05

**valid** = [-inf, inf]

**variance** = Binomial

**weights (mu)**

Weights for IRLS step.

\[ w = \frac{1}{\text{link}'(mu)^2 * \text{variance}(mu)} \]

**INPUTS:**
- mu – mean parameter in exponential family

**OUTPUTS:**
- w – weights used in WLS step of GLM/GAM fit
54.2.2 Family

class nipy.algorithms.statistics.models.family.family.Family(link, variance)

Bases: object

A class to model one-parameter exponential families.

INPUTS: link – a Link instance variance – a variance function (models means as a function of mean)

Methods

---

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviance(Y, mu[, scale])</td>
<td>Deviance of (Y, mu) pair. Deviance is usually defined as the difference</td>
</tr>
<tr>
<td>devresid(Y, mu)</td>
<td>The deviance residuals, defined as the residuals in the deviance.</td>
</tr>
<tr>
<td>fitted(eta)</td>
<td>Fitted values based on linear predictors eta.</td>
</tr>
<tr>
<td>predict(mu)</td>
<td>Linear predictors based on given mu values.</td>
</tr>
<tr>
<td>weights(mu)</td>
<td>Weights for IRLS step.</td>
</tr>
</tbody>
</table>

__init__(link, variance)

deviance(Y, mu, scale=1.0)

Deviance of (Y, mu) pair. Deviance is usually defined as

\[
\text{DEV} = \frac{\text{SUM}_i -2 \log \text{Likelihood}(Y_i,\mu_i) + 2 \log \text{Likelihood}(\mu_i,\mu_i)}{\text{scale}}
\]

INPUTS: Y – response variable mu – mean parameter scale – optional scale in denominator of deviance

OUTPUTS: dev  dev = DEV, as described above

devresid(Y, mu)

The deviance residuals, defined as the residuals in the deviance.

Without knowing the link, they default to Pearson residuals

\[
\text{resid}_P = (Y - \mu) \times \sqrt{\text{weight}(\mu)}
\]

INPUTS: Y – response variable mu – mean parameter

OUTPUTS: resid  resid = deviance residuals

fitted(eta)

Fitted values based on linear predictors eta.

INPUTS:

eta – values of linear predictors, say, X beta in a generalized linear model.

OUTPUTS: mu  mu = link.inverse(eta), mean parameter based on eta

link

links = []

predict(mu)

Linear predictors based on given mu values.

INPUTS: mu – mean parameter of one-parameter exponential family

OUTPUTS: eta

eta – link(mu), linear predictors, based on mean parameters mu
tol = 1e-05
valid = [-inf, inf]
weights (mu)
   Weights for IRLS step.
   w = 1 / (link'(mu)**2 * variance(mu))
   INPUTS: mu – mean parameter in exponential family
   OUTPUTS: w – weights used in WLS step of GLM/GAM fit

54.2.3 Gamma
class nipy.algorithms.statistics.models.family.family.Gamma (link=<nipy.algorithms.statistics.models.family.links.Power object at 0x5225090>):
   Bases: nipy.algorithms.statistics.models.family.family.Family
   Gamma exponential family.
   INPUTS: link – a Link instance
   BUGS: no deviance residuals?

Methods

   deviance(Y, mu[, scale]) Deviance of (Y, mu) pair. Deviance is usually defined
   devresid(Y, mu) The deviance residuals, defined as the residuals in the deviance.
   fitted(eta) Fitted values based on linear predictors eta.
   predict(mu) Linear predictors based on given mu values.
   variance Power variance function:
   weights(mu) Weights for IRLS step.

   __init__ (link=<nipy.algorithms.statistics.models.family.links.Power object at 0x5225090>)
   deviance (Y, mu, scale=1.0)
      Deviance of (Y, mu) pair. Deviance is usually defined as the difference
      DEV = (SUM_i -2 log Likelihood(Y_i, mu_i) + 2 log Likelihood(mu_i, mu_i)) / scale
      INPUTS: Y – response variable mu – mean parameter scale – optional scale in denominator of deviance
      OUTPUTS: dev dev – DEV, as described above
   devresid (Y, mu)
      The deviance residuals, defined as the residuals in the deviance.
      Without knowing the link, they default to Pearson residuals
      resid_P = (Y - mu) * sqrt(weight(mu))
      INPUTS: Y – response variable mu – mean parameter
      OUTPUTS: resid resid – deviance residuals
   fitted (eta)
      Fitted values based on linear predictors eta.
      INPUTS:
eta – values of linear predictors, say, X beta in a generalized linear model.

OUTPUTS: mu
mu – link.inverse(eta), mean parameter based on eta

link

links = [nipy.algorithms.statistics.models.family.links.Log object at 0x52250d0], <nipy.algorithms.statistics.models.family.links.Power object at 0x5225090>, <nipy.algorithms.statistics.models.family.links.Power object at 0x5217f50>

predict (mu)
Linear predictors based on given mu values.

INPUTS: mu – mean parameter of one-parameter exponential family

OUTPUTS: eta
eta – link(mu), linear predictors, based on mean parameters mu

tol = 1e-05
valid = [-inf, inf]

variance = <nipy.algorithms.statistics.models.family.varfuncs.Power object at 0x5225690>

weights (mu)
Weights for IRLS step.

w = 1 / (link'(mu)**2 * variance(mu))

INPUTS: mu – mean parameter in exponential family

OUTPUTS: w – weights used in WLS step of GLM/GAM fit

54.2.4 Gaussian

class nipy.algorithms.statistics.models.family.family.Gaussian (link=<nipy.algorithms.statistics.models.family.links.Power object at 0x5225090>)

Bases: nipy.algorithms.statistics.models.family.family.Family

Gaussian exponential family.

INPUTS: link – a Link instance

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviance</td>
<td>Deviance of (Y,mu) pair. Deviance is usually defined</td>
</tr>
<tr>
<td>devresid</td>
<td>Gaussian deviance residual</td>
</tr>
<tr>
<td>fitted</td>
<td>Fitted values based on linear predictors eta.</td>
</tr>
<tr>
<td>predict</td>
<td>Linear predictors based on given mu values.</td>
</tr>
<tr>
<td>variance</td>
<td>Variance function that relates the variance of a random variable to its mean.</td>
</tr>
<tr>
<td>weights</td>
<td>Weights for IRLS step</td>
</tr>
</tbody>
</table>

__init__ (link=<nipy.algorithms.statistics.models.family.links.Power object at 0x5225090>)

deviance (Y, mu, scale=1.0)
Deviance of (Y,mu) pair. Deviance is usually defined as the difference

DEV = (SUM_i -2 log Likelihood(Y_i,mu_i) + 2 log Likelihood(mu_i,mu_i)) / scale

INPUTS: Y – response variable mu – mean parameter scale – optional scale in denominator of deviance
Neuroimaging in Python Documentation, Release 0.3.0

**OUTPUTS: dev** dev – DEV, as described above

```
devresid(Y, mu, scale=1.0)
```

Gaussian deviance residual

**INPUTS:**
- Y – response variable
- mu – mean parameter
- scale – optional scale in denominator (after taking $\sqrt{\cdot}$)

**OUTPUTS:** resid resid – deviance residuals

```
fitted(eta)
```

Fitted values based on linear predictors eta.

**INPUTS:**
- eta – values of linear predictors, say, $X\beta$ in a generalized linear model.

**OUTPUTS:** mu mu – link.inverse(eta), mean parameter based on eta

```
link
```

**links** = [nipy.algorithms.statistics.models.family.links.Log object at 0x52250d0], <nipy.algorithms.statistics.models.family.family.Power object at 0x5225090>, <nipy.algorithms.statistics.models.family.family.Power object at 0x5217f50>

```
predict(mu)
```

Linear predictors based on given mu values.

**INPUTS:**
- mu – mean parameter of one-parameter exponential family

**OUTPUTS:** eta
- eta – link(mu), linear predictors, based on mean parameters mu

```
tol = 1e-05
valid = [-inf, inf]
variance = <nipy.algorithms.statistics.models.family.family.varfuncs.VarianceFunction object at 0x52254d0>
weights(mu)
```

Weights for IRLS step.

```
w = 1 / (link'(mu)**2 * variance(mu))
```

**INPUTS:**
- mu – mean parameter in exponential family

**OUTPUTS:** w – weights used in WLS step of GLM/GAM fit

### 54.2.5 InverseGaussian

```
class nipy.algorithms.statistics.models.family.family.InverseGaussian(link=nipy.algorithms.statistics.models.family.family.Power object at 0x5225090)
```

**Bases:** nipy.algorithms.statistics.models.family.family.Family

InverseGaussian exponential family.

**INPUTS:**
- link – a Link instance
- n – number of trials for Binomial

**Methods**

```
deviance(Y, mu[, scale]) Deviance of (Y,mu) pair. Deviance is usually defined
devresid(Y, mu) The deviance residuals, defined as the residuals in the deviance.
```

Continued on next page
**Table 54.5 – continued from previous page**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fitted(eta)</code></td>
<td>Fitted values based on linear predictors eta.</td>
</tr>
<tr>
<td><code>predict(mu)</code></td>
<td>Linear predictors based on given mu values.</td>
</tr>
<tr>
<td><code>variance</code></td>
<td>Power variance function:</td>
</tr>
<tr>
<td><code>weights(mu)</code></td>
<td>Weights for IRLS step.</td>
</tr>
</tbody>
</table>

**__init__**

```
link=<nipy.algorithms.statistics.models.family.links.Power object at 0x5225090>
```

**deviance** *(Y, mu, scale=1.0)*

Deviance of (Y,mu) pair. Deviance is usually defined as the difference

\[
DEV = \frac{\text{SUM}_i 2 \log \text{Likelihood}(Y_i,\mu_i) + 2 \log \text{Likelihood}(\mu_i,\mu_i)}{\text{scale}}
\]

**INPUTS:** Y – response variable mu – mean parameter scale – optional scale in denominator of deviance

**OUTPUTS:** dev  dev – DEV, as described above

**devresid** *(Y, mu)*

The deviance residuals, defined as the residuals in the deviance.

Without knowing the link, they default to Pearson residuals

\[
\text{resid}_P = (Y - \mu) * \sqrt{\text{weight}(\mu)}
\]

**INPUTS:** Y – response variable mu – mean parameter

**OUTPUTS:** resid  resid – deviance residuals

**fitted** *(eta)*

Fitted values based on linear predictors eta.

**INPUTS:**

eta – values of linear predictors, say, X beta in a generalized linear model.

**OUTPUTS:** mu  mu – link.inverse(eta), mean parameter based on eta

**link**

```
links = [<nipy.algorithms.statistics.models.family.links.Power object at 0x5225050>, <nipy.algorithms.statistics.models.family.links.Log object at 0x52250d0>]
```

**predict** *(mu)*

Linear predictors based on given mu values.

**INPUTS:** mu – mean parameter of one-parameter exponential family

**OUTPUTS:** eta  eta – link(mu), linear predictors, based on mean parameters mu

**tol = 1e-05**

**valid = [-inf, inf]**

**variance = <nipy.algorithms.statistics.models.family.varfuncs.Power object at 0x52256d0>**

**weights** *(mu)*

Weights for IRLS step.

\[
w = \frac{1}{\text{link}'(\mu)^2 \cdot \text{variance}(\mu)}
\]

**INPUTS:** mu – mean parameter in exponential family

**OUTPUTS:** w – weights used in WLS step of GLM/GAM fit
54.2.6 Poisson

class nipy.algorithms.statistics.models.family.family.Poisson(link=nipy.algorithms.statistics.models.family.family.Log object at 0x52250d0)

Bases: nipy.algorithms.statistics.models.family.family.Family

Poisson exponential family.

INPUTS: link – a Link instance

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>deviance(Y, mu[, scale])</td>
<td>Deviance of (Y, mu) pair. Deviance is usually defined</td>
</tr>
<tr>
<td>devresid(Y, mu)</td>
<td>Poisson deviance residual</td>
</tr>
<tr>
<td>fitted(eta)</td>
<td>Fitted values based on linear predictors eta.</td>
</tr>
<tr>
<td>predict(mu)</td>
<td>Linear predictors based on given mu values.</td>
</tr>
<tr>
<td>variance</td>
<td>Power variance function:</td>
</tr>
<tr>
<td>weights(mu)</td>
<td>Weights for IRLS step.</td>
</tr>
</tbody>
</table>

__init__ (link=nipy.algorithms.statistics.models.family.family.Log object at 0x52250d0)

deviance (Y, mu, scale=1.0)

Deviance of (Y, mu) pair. Deviance is usually defined as the difference

DEV = (SUM_i -2 log Likelihood(Y_i,mu_i) + 2 log Likelihood(mu_i,mu_i)) / scale

INPUTS: Y – response variable mu – mean parameter scale – optional scale in denominator of deviance

OUTPUTS: dev dev – DEV, as described above

devresid(Y, mu)

Poisson deviance residual

INPUTS: Y – response variable mu – mean parameter

OUTPUTS: resid resid – deviance residuals

fitted(eta)

Fitted values based on linear predictors eta.

INPUTS:

eta – values of linear predictors, say, X beta in a generalized linear model.

OUTPUTS: mu mu – link.inverse(eta), mean parameter based on eta

link

links = [nipy.algorithms.statistics.models.family.links.Log object at 0x52250d0>, nipy.algorithms.statistics.models.family.links.Power object at 0x5217fd0>

predict(mu)

Linear predictors based on given mu values.

INPUTS: mu – mean parameter of one-parameter exponential family

OUTPUTS: eta

eta – link(mu), linear predictors, based on mean parameters mu

tol = 1e-05
valid = [0, inf]

variance = <nipy.algorithms.statistics.models.family.varfuncs.Power object at 0x5225610>

weights (mu)

Weights for IRLS step.

w = 1 / (link'(mu)**2 * variance(mu))

INPUTS:  mu – mean parameter in exponential family

OUTPUTS:  w – weights used in WLS step of GLM/GAM fit
55.1 Module: `algorithms.statistics.models.family.links`

Inheritance diagram for `nipy.algorithms.statistics.models.family.links`:

```
family.links.Logit
   /       \
  /         \nfamily.links.CDFLink family.links.CLogLog
```

55.2 Classes

55.2.1 CDFLink

```python
class nipy.algorithms.statistics.models.family.links.CDFLink
```

The use the CDF of a scpy.stats distribution as a link function:

\[ g(x) = \text{dbn.}
```
Neuroimaging in Python Documentation, Release 0.3.0

__call__ (p)  CDF link

_clean (p)  Clip logistic values to range (tol, 1-tol)
_deriv (p)  Derivative of CDF link
_initialize (Y)  Derivative of CDF link

__init__ (dbn=<scipy.stats.distributions.norm_gen object at 0x3d684d0>)

_clean (p)
  Clip logistic values to range (tol, 1-tol)
  INPUTS:  p – probabilities
  OUTPUTS:  pclip  pclip – clipped probabilities

deriv (p)
  Derivative of CDF link
  g(p) = 1/self.dbn.pdf(self.dbn.ppf(p))
  INPUTS:  x – mean parameters
  OUTPUTS:  z  z – derivative of CDF transform of x

_initialize (Y)

_inverse (z)
  Derivative of CDF link
  g(z) = self.dbn.cdf(z)
  INPUTS:  z – linear predictors in GLM
  OUTPUTS:  p  p – inverse of CDF link of z

tol = 1e-10

55.2.2 CLogLog

class nipy.algorithms.statistics.models.family.links.CLogLog
Bases: nipy.algorithms.statistics.models.family.links.Logit

The complementary log-log transform as a link function:
  g(x) = log(-log(x))

Methods

__call__ (p)  C-Log-Log transform
_clean (p)  Clip logistic values to range (tol, 1-tol)
_deriv (p)  Derivative of C-Log-Log transform
_initialize (Y)  Derivative of C-Log-Log transform

_inverse (z)  Inverse of C-Log-Log transform

__init__ ()
  x.__init__(...) initializes x; see help(type(x)) for signature
clean \((p)\)
Clips logistic values to range \((\text{tol}, 1-\text{tol})\)

**INPUTS:** \(p\) – probabilities

**OUTPUTS:** \(p_{\text{clip}}\) \(p_{\text{clip}}\) – clipped probabilities

deriv \((p)\)
Derivative of C-Log-Log transform
\[ g(p) = -\frac{1}{\log(p) \cdot p} \]

**INPUTS:** \(p\) – mean parameters

**OUTPUTS:** \(z\) \(z\) – \(-\frac{1}{\log(p) \cdot p}\)

initialize \((Y)\)

inverse \((z)\)
Inverse of C-Log-Log transform
\[ g(z) = \exp(-\exp(z)) \]

**INPUTS:** \(z\) – linear predictor scale

**OUTPUTS:** \(p\) \(p\) – mean parameters

tol = 1e-10

55.2.3 Link

class nipy.algorithms.statistics.models.family.links.Link
Bases: object

A generic link function for one-parameter exponential family, with call, inverse and deriv methods.

Methods

```python
__call__(p)
deriv(p)
initialize(Y)
inverse(z)
```

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature
deriv \((p)\)
initialize \((Y)\)
inverse \((z)\)

55.2.4 Log

class nipy.algorithms.statistics.models.family.links.Log
Bases: nipy.algorithms.statistics.models.family.links.Link

The log transform as a link function:
g(x) = log(x)

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong></td>
<td>Log transform</td>
</tr>
<tr>
<td>clean(x)</td>
<td></td>
</tr>
<tr>
<td>deriv(x)</td>
<td>Derivative of log transform</td>
</tr>
<tr>
<td>initialize(Y)</td>
<td></td>
</tr>
<tr>
<td>inverse(z)</td>
<td>Inverse of log transform</td>
</tr>
</tbody>
</table>

__init__(...) initializes x; see help(type(x)) for signature

clean(x)

deriv(x)

Derivative of log transform

g(x) = 1/x

INPUTS:  x – mean parameters

OUTPUTS: z  z – derivative of log transform of x

initialize(Y)

inverse(z)

Inverse of log transform

g(x) = exp(x)

INPUTS:  z – linear predictors in GLM

OUTPUTS: x  x – exp(z)

tol = 1e-10

55.2.5 Logit

class nipy.algorithms.statistics.models.family.links.Logit

Bases: nipy.algorithms.statistics.models.family.links.Link

The logit transform as a link function:

g'(x) = 1 / (x * (1 - x))  g^(-1)(x) = exp(x)/(1 + exp(x))

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong></td>
<td>Logit transform</td>
</tr>
<tr>
<td>clean(p)</td>
<td>Clip logistic values to range (tol, 1-tol)</td>
</tr>
<tr>
<td>deriv(p)</td>
<td>Derivative of logit transform</td>
</tr>
<tr>
<td>initialize(Y)</td>
<td></td>
</tr>
<tr>
<td>inverse(z)</td>
<td>Inverse logit transform</td>
</tr>
</tbody>
</table>
__init__(...) initializes x; see help(type(x)) for signature

**clean**(p)
Clip logistic values to range (tol, 1-tol)

**INPUTS:** p – probabilities

**OUTPUTS:** pclip  pclip – clipped probabilities

**deriv**(p)
Derivative of logit transform

\[ g(p) = \frac{1}{p \cdot (1 - p)} \]

**INPUTS:** p – probabilities

**OUTPUTS:** y  y – derivative of logit transform of p

**initialize**(Y)

**inverse**(z)
Inverse logit transform

\[ h(z) = \frac{\exp(z)}{1 + \exp(z)} \]

**INPUTS:** z – logit transform of p

**OUTPUTS:** p  p – probabilities

**tol** = 1e-10

### 55.2.6 Power

**class nipy.algorithms.statistics.models.family.links.Power**(power=1.0)
**Bases:** nipy.algorithms.statistics.models.family.links.Link

The power transform as a link function:

\[ g(x) = x^{power} \]

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>call</strong>(x)</td>
<td>Power transform</td>
</tr>
<tr>
<td>deriv(x)</td>
<td>Derivative of power transform</td>
</tr>
<tr>
<td>initialize(Y)</td>
<td>Inverse of power transform</td>
</tr>
<tr>
<td>inverse(z)</td>
<td>Inverse of power transform</td>
</tr>
</tbody>
</table>

**__init__(power=1.0)**

**deriv**(x)
Derivative of power transform

\[ g(x) = self.power \cdot x^{(self.power - 1)} \]

**INPUTS:** x – mean parameters

**OUTPUTS:** z  z – derivative of power transform of x

**initialize**(Y)

### 55.2. Classes
**inverse**($z$)

Inverse of power transform

$g(x) = x^{**(1/self.power)}$

**INPUTS:** $z$ – linear predictors in GLM

**OUTPUTS:** $x$  $x$ – mean parameters
56.1 Module: algorithms.statistics.models.family.varfuncs

Inheritance diagram for nipy.algorithms.statistics.models.family.varfuncs:

- family.varfuncs.Binomial
- family.varfuncs.VarianceFunction
- family.varfuncs.Power

56.2 Classes

56.2.1 Binomial

class nipy.algorithms.statistics.models.family.varfuncs.Binomial(n=1):
    Bases: object
    Binomial variance function
    p = mu / n; V(mu) = p * (1 - p) * n
    INPUTS: n – number of trials in Binomial

    Methods

    __call__(mu)  Binomial variance function
    clean(p)
56.2.2 Power

class nipy.algorithms.statistics.models.family.varfuncs.Power(power=1.0)
    Bases: object
    Power variance function:
    \[ V(\mu) = |\mu|^{\text{power}} \]
    INPUTS: power – exponent used in power variance function

    Methods

    __call__(mu)  Power variance function
    __init__(power=1.0)

56.2.3 VarianceFunction

class nipy.algorithms.statistics.models.family.varfuncs.VarianceFunction
    Bases: object
    Variance function that relates the variance of a random variable to its mean. Defaults to 1.

    Methods

    __call__(mu)  Default variance function
    __init__() x.__init__(...) initializes x; see help(type(x)) for signature
ALGORITHMS.STATISTICS.MODELS.GLM

57.1 Module: algorithms.statistics.models.glm

Inheritance diagram for nipy.algorithms.statistics.models.glm:

57.1.1 General linear models

57.2 Model

class nipy.algorithms.statistics.models.glm.Model(design, 
    family=nipy.algorithms.statistics.models.family.family.Gaussian
object at 0x52259d0>)

Bases: nipy.algorithms.statistics.models.regression.WLSModel

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cont([tol])</td>
<td>Continue iterating, or has convergence been obtained?</td>
</tr>
<tr>
<td>deviance([Y, results, scale])</td>
<td>Return (unnormalized) log-likelihood for GLM.</td>
</tr>
<tr>
<td>estimate_scale([Y, results])</td>
<td>Return Pearson’s X^2 estimate of scale.</td>
</tr>
<tr>
<td>fit(Y)</td>
<td></td>
</tr>
<tr>
<td>has_intercept()</td>
<td>Check if column of 1s is in column space of design</td>
</tr>
<tr>
<td>information(beta[, nuisance])</td>
<td>Returns the information matrix at (beta, Y, nuisance).</td>
</tr>
<tr>
<td>initialize(design)</td>
<td></td>
</tr>
<tr>
<td>logl(beta, Y[, nuisance])</td>
<td>Returns the value of the loglikelihood function at beta.</td>
</tr>
<tr>
<td>next()</td>
<td></td>
</tr>
<tr>
<td>predict([design])</td>
<td>After a model has been fit, results are (assumed to be) stored</td>
</tr>
<tr>
<td>rank()</td>
<td>Compute rank of design matrix</td>
</tr>
<tr>
<td>score(beta, Y[, nuisance])</td>
<td>Gradient of the loglikelihood function at (beta, Y, nuisance).</td>
</tr>
<tr>
<td>whiten(X)</td>
<td>Whitener for WLS model, multiplies by sqrt(self.weights)</td>
</tr>
</tbody>
</table>
__init__ (design, family=<nipy.algorithms.statistics.models.family.family.Gaussian object at 0x52259d0>)
cont (tol=1e-05)
    Continue iterating, or has convergence been obtained?
deviance (Y=None, results=None, scale=1.0)
    Return (unnormalized) log-likelihood for GLM.
    Note that self.scale is interpreted as a variance in old_model, so we divide the residuals by its sqrt.
estimate_scale (Y=None, results=None)
    Return Pearson’s X^2 estimate of scale.
fit (Y)
static has_intercept ()
    Check if column of 1s is in column space of design
information (beta, nuisance=None)
    Returns the information matrix at (beta, Y, nuisance).
    See logL for details.

Parameters

beta : ndarray
    The parameter estimates. Must be of length df_model.

nuisance : dict
    A dict with key 'sigma', which is an estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as \sum ((Y - X*beta)**2) / n where n=Y.shape[0], X=self.design.

Returns

info : array
    The information matrix, the negative of the inverse of the Hessian of the of the loglikelihood function evaluated at (theta, Y, nuisance).

initialize (design)

logL (beta, Y, nuisance=None)
    Returns the value of the loglikelihood function at beta.
    Given the whitened design matrix, the loglikelihood is evaluated at the parameter vector, beta, for the dependent variable, Y and the nuisance parameter, sigma.

Parameters

beta : ndarray
    The parameter estimates. Must be of length df_model.

Y : ndarray
    The dependent variable

nuisance : dict, optional
    A dict with key 'sigma', which is an optional estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as \sum ((Y - X*beta)**2) / n, where n=Y.shape[0], X=self.design.

Returns

loglf : float
    The value of the loglikelihood function.
Notes

The log-Likelihood Function is defined as

$$\ell(\beta, \sigma, Y) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{\|Y - X\beta\|^2}{2\sigma^2}$$

The parameter $\sigma$ above is what is sometimes referred to as a nuisance parameter. That is, the likelihood is considered as a function of $\beta$, but to evaluate it, a value of $\sigma$ is needed.

If $\sigma$ is not provided, then its maximum likelihood estimate:

$$\hat{\sigma}(\beta) = \frac{\text{SSE}(\beta)}{n}$$

is plugged in. This likelihood is now a function of only $\beta$ and is technically referred to as a profile-likelihood.

References

[R1]

```python
next()
niter = 10
predict(design=None)
```

After a model has been fit, results are (assumed to be) stored in self.results, which itself should have a predict method.

```python
static rank()
```

Compute rank of design matrix

```python
score(beta, Y, nuisance=None)
```

Gradient of the loglikelihood function at (beta, Y, nuisance).

The graident of the loglikelihood function at (beta, Y, nuisance) is the score function.

See logL() for details.

**Parameters**

- `beta` : ndarray
  
  The parameter estimates. Must be of length df_model.

- `Y` : ndarray
  
  The dependent variable.

- `nuisance` : dict, optional
  
  A dict with key 'sigma', which is an optional estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as $\text{sum}((Y - X*beta)^2) / n$, where $n=Y.shape[0]$, $X=self.design$.

**Returns**

The gradient of the loglikelihood function.

```python
whiten(X)
```

Whitener for WLS model, multiplies by sqrt(self.weights)
58.1 Module: algorithms.statistics.models.model

Inheritance diagram for nipy.algorithms.statistics.models.model:

```python
models.model.LikelihoodModelResults

models.model.FContrastResults

models.model.TContrastResults

models.model.Model

models.model.LikelihoodModel
```

58.2 Classes

58.2.1 FContrastResults

```python
class nipy.algorithms.statistics.models.model.FContrastResults(effect, covariance, F, df_num, df_den=None)
```

Bases: object

Results from an F contrast of coefficients in a parametric model.

The class does nothing, it is a container for the results from F contrasts, and returns the F-statistics when np.asarray is called.

```python
__init__(effect, covariance, F, df_num, df_den=None)
```
58.2.2 LikelihoodModel

class nipy.algorithms.statistics.models.model.LikelihoodModel
    Bases: nipy.algorithms.statistics.models.model.Model

Methods

    fit() Fit a model to data.
    information(theta[, nuisance]) Fisher information matrix
    initialize() Initialize (possibly re-initialize) a Model instance.
    logL(theta, Y[, nuisance]) Log-likelihood of model.
    predict([design]) After a model has been fit, results are (assumed to be) stored
    score(theta, Y[, nuisance]) Gradient of logL with respect to theta.

    __init__()  
    fit() Fit a model to data.
    information(theta, nuisance=None) Fisher information matrix
        The inverse of the expected value of - d^2 logL / dtheta^2.
    initialize() Initialize (possibly re-initialize) a Model instance.
        For instance, the design matrix of a linear model may change and some things must be recomputed.
    logL(theta, Y, nuisance=None) Log-likelihood of model.
    predict(design=None) After a model has been fit, results are (assumed to be) stored in self.results, which itself should have a
        predict method.
    score(theta, Y, nuisance=None) Gradient of logL with respect to theta.
        This is the score function of the model

58.2.3 LikelihoodModelResults

class nipy.algorithms.statistics.models.model.LikelihoodModelResults(theta, Y, model, cov=None, dispersion=1.0, nuisance=None, rank=None)

Bases: object
Class to contain results from likelihood models
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AIC()</td>
<td>Akaike Information Criterion</td>
</tr>
<tr>
<td>BIC()</td>
<td>Schwarz’s Bayesian Information Criterion</td>
</tr>
<tr>
<td>Fcontrast(matrix[, dispersion, invcov])</td>
<td>Compute an Fcontrast for a contrast matrix matrix.</td>
</tr>
<tr>
<td>Tcontrast(matrix[, store, dispersion])</td>
<td>Compute a Tcontrast for a row vector matrix</td>
</tr>
<tr>
<td>conf_int([alpha, cols, dispersion])</td>
<td>The confidence interval of the specified theta estimates.</td>
</tr>
<tr>
<td>logL()</td>
<td>The maximized log-likelihood</td>
</tr>
<tr>
<td>t([column])</td>
<td>Return the (Wald) t-statistic for a given parameter estimate.</td>
</tr>
<tr>
<td>vcov([matrix, column, dispersion, other])</td>
<td>Variance/covariance matrix of linear contrast</td>
</tr>
</tbody>
</table>

__init__ (theta, Y, model, cov=None, dispersion=1.0, nuisance=None, rank=None)

Set up results structure

Parameters

- **theta** : ndarray
  parameter estimates from estimated model
- **Y** : ndarray
  data
- **model** : LikelihoodModel instance
  model used to generate fit
- **cov** : None or ndarray, optional
  covariance of thetas
- **dispersion** : scalar, optional
  multiplicative factor in front of cov
- **nuisance** : None or ndarray
  parameter estimates needed to compute logL
- **rank** : None or scalar
  rank of the model. If rank is not None, it is used for df_model instead of the usual counting of parameters.

Notes

The covariance of thetas is given by:

dispersion * cov

For (some subset of models) dispersion will typically be the mean square error from the estimated model (sigma^2)

static AIC ()
Akaiake Information Criterion

static BIC ()
Schwarz’s Bayesian Information Criterion
Fcontrast(matrix, dispersion=None, invcov=None)
Compute an Fcontrast for a contrast matrix matrix.
Here, matrix M is assumed to be non-singular. More precisely

\[ MpXpX' M' \]

is assumed invertible. Here, \( pX \) is the generalized inverse of the design matrix of the model. There can be problems in non-OLS models where the rank of the covariance of the noise is not full.

See the contrast module to see how to specify contrasts. In particular, the matrices from these contrasts will always be non-singular in the sense above.

**Parameters**
- **matrix**: 1D array-like
  contrast matrix
- **dispersion**: None or float, optional
  If None, use self.dispersion
- **invcov**: None or array, optional
  Known inverse of variance covariance matrix. If None, calculate this matrix.

**Returns**
- **f_res**: FContrastResults instance
  with attributes F, df_den, df_num

**Notes**
For F contrasts, we now specify an effect and covariance

Tcontrast(matrix, store=('t', 'effect', 'sd'), dispersion=None)
Compute a Tcontrast for a row vector matrix
To get the t-statistic for a single column, use the ‘t’ method.

**Parameters**
- **matrix**: 1D array-like
  contrast matrix
- **store**: sequence, optional
  components of t to store in results output object. Defaults to all components (‘t’, ‘effect’, ‘sd’).
- **dispersion**: None or float, optional

**Returns**
- **res**: TContrastResults object

conf_int(alpha=0.05, cols=None, dispersion=None)
The confidence interval of the specified theta estimates.

**Parameters**
- **alpha**: float, optional
  The alpha level for the confidence interval. ie., alpha = .05 returns a 95% confidence interval.
- **cols**: tuple, optional
  cols specifies which confidence intervals to return
- **dispersion**: None or scalar
scale factor for the variance / covariance (see class docstring and `vcov` method docstring)

**Returns**

`cis : ndarray`

`cis` is shape `(len(cols), 2)` where each row contains [lower, upper] for the given entry in `cols`

**Notes**

Confidence intervals are two-tailed. TODO: tails : string, optional

`tails` can be “two”, “upper”, or “lower”

**Examples**

```python
>>> from numpy.random import standard_normal as stan
>>> from nipy.algorithms.statistics.models.regression import OLSModel
>>> x = np.hstack((stan((30,1)),stan((30,1)),stan((30,1))))
>>> beta=np.array([3.25, 1.5, 7.0])
>>> y = np.dot(x,beta) + stan((30))
>>> model = OLSModel(x).fit(y)
>>> confidence_intervals = model.conf_int(cols=(1,2))
```

**static `logL()`**

The maximized log-likelihood

**t** (`column=None`)

Return the (Wald) t-statistic for a given parameter estimate.

Use `Tcontrast` for more complicated (Wald) t-statistics.

**vcov** (`matrix=None, column=None, dispersion=None, other=None`)

Variance/covariance matrix of linear contrast

**Parameters**

`matrix: (dim, self.theta.shape[0]) array, optional`:

numerical contrast specification, where `dim` refers to the ‘dimension’ of the contrast i.e.
1 for t contrasts, 1 or more for F contrasts.

`column: int, optional`:

alternative way of specifying contrasts (column index)

`dispersion: float or (n_voxels,) array, optional`:

value(s) for the dispersion parameters

`other: (dim, self.theta.shape[0]) array, optional`:

alternative contrast specification (?)

**Returns**

`cov: (dim, dim) or (n_voxels, dim, dim) array`

the estimated covariance matrix/matrices

Returns the variance/covariance matrix of a linear contrast of the
estimates of theta, multiplied by ‘dispersion’ which will often be an estimate of ‘dispersion’, like, sigma^2.

The covariance of interest is either specified as a (set of) column(s) :
or a matrix.

58.2.4 Model

class nipy.algorithms.statistics.models.model.Model
    Bases: object
    A (predictive) statistical model.
    The class Model itself does nothing but lays out the methods expected of any subclass.

Methods

    fit() Fit a model to data.
    initialize() Initialize (possibly re-initialize) a Model instance.
    predict([design]) After a model has been fit, results are (assumed to be) stored

__init__()

fit()
    Fit a model to data.

initialize()
    Initialize (possibly re-initialize) a Model instance.
    For instance, the design matrix of a linear model may change and some things must be recomputed.

predict([design=None])
    After a model has been fit, results are (assumed to be) stored in self.results, which itself should have a
    predict method.

58.2.5 TContrastResults

class nipy.algorithms.statistics.models.model.TContrastResults(t, sd, effect, df_den=None)
    Bases: object
    Results from a t contrast of coefficients in a parametric model.
    The class does nothing, it is a container for the results from T contrasts, and returns the T-statistics when
    np.asarray is called.

__init__((t, sd, effect, df_den=None))
59.1 Module: `algorithms.statistics.models.nlsmodel`

Inheritance diagram for `nipy.algorithms.statistics.models.nlsmodel`:

```
models.model.Model  <--> models.nlsmodel.NLSModel
```

Non-linear least squares model

59.2 NLSModel

```python
class NLSModel(Y, design, f, grad, theta, niter=10):
    Bases: nipy.algorithms.statistics.models.model.Model
    Class representing a simple nonlinear least squares model.
```

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSE()</td>
<td>Sum of squares error.</td>
</tr>
<tr>
<td>fit()</td>
<td>Fit a model to data.</td>
</tr>
<tr>
<td>getZ()</td>
<td>Set Z into self</td>
</tr>
<tr>
<td>getomega()</td>
<td>Set omega into self</td>
</tr>
<tr>
<td>initialize()</td>
<td>Initialize (possibly re-initialize) a Model instance.</td>
</tr>
<tr>
<td>next()</td>
<td>Do an iteration of fit</td>
</tr>
<tr>
<td>predict([design])</td>
<td>Get predicted values for design or self.design</td>
</tr>
</tbody>
</table>

```python
__init__(Y, design, f, grad, theta, niter=10)
```

Initialize non-linear model instance

**Parameters**

- `Y`: ndarray
the data in the NLS model

**design** : ndarray

the design matrix, \(X\)

**f** : callable

the map between the (linear parameters (in the design matrix) and the nonlinear parameters (\(\theta\))) and the predicted data. \(f\) accepts the design matrix and the parameters (\(\theta\)) as input, and returns the predicted data at that design.

**grad** : callable

the gradient of \(f\), this should be a function of an \(nxp\) design matrix \(X\) and \(qx1\) vector \(\theta\) that returns an \(nxq\) matrix \(\frac{df_i}{d\theta_j}\) where:

\[
f_i(\theta) = f(X[i], \theta)
\]

is the nonlinear response function for the \(i\)-th instance in the model.

**theta** : array

parameters

**niter** : int

number of iterations

**SSE()**

Sum of squares error.

Returns **sse** : float :

sum of squared residuals

**fit()**

Fit a model to data.

**getZ()**

Set \(Z\) into \(self\)

Returns **None** :

**getomega()**

Set \(\omega\) into \(self\)

Returns **None** :

**initialize()**

Initialize (possibly re-initialize) a Model instance.

For instance, the design matrix of a linear model may change and some things must be recomputed.

**next()**

Do an iteration of fit

Returns **None** :

**predict** *(design=None)*

Get predicted values for \(design\) or \(self.design\)

Parameters **design** : None or array, optional

\(design\) at which to predict data. If None (the default) then use the initial \(self.design\)

Returns **y_predicted** : array
predicted data at given (or initial) design
60.1 Module: algorithms.statistics.models.regression

Inheritance diagram for nipy.algorithms.statistics.models.regression:

This module implements some standard regression models: OLS and WLS models, as well as an AR(p) regression model.

Models are specified with a design matrix and are fit using their ‘fit’ method.

Subclasses that have more complicated covariance matrices should write over the ‘whiten’ method as the fit method prewhitens the response by calling ‘whiten’.

General reference for regression models:


60.2 Classes

60.2.1 AREstimator

```python
class nipy.algorithms.statistics.models.regression.AREstimator(model, p=1)
    Bases: object

    A class to estimate AR(p) coefficients from residuals
```
Methods
_call_(results)  Calculate AR(p) coefficients from results.'residuals'

__init__(model, p=1)
Bias-correcting AR estimation class

Parameters
model : OLSModel instance
A models.regression.OLSmodel instance, where model has attribute design
p : int, optional
Order of AR(p) noise

60.2.2 ARModel
class nipy.algorithms.statistics.models.regression.ARMModel(design, rho)
Bases: nipy.algorithms.statistics.models.regression.OLSModel

A regression model with an AR(p) covariance structure.

In terms of a LikelihoodModel, the parameters are beta, the usual regression parameters, and sigma, a scalar
nuisance parameter that shows up as multiplier in front of the AR(p) covariance.

The linear autoregressive process of order p–AR(p)–is defined as: TODO

Examples

```python
>>> from nipy.algorithms.statistics.api import Term, Formula
>>> data = np.rec.fromarrays(([1, 3, 4, 5, 8, 10, 9], range(1, 8)),
...   names=('Y', 'X'))
>>> f = Formula([Term("X"), 1])
>>> dmtx = f.design(data, return_float=True)
>>> model = ARModel(dmtx, 2)

We go through the model.iterative_fit procedure long-hand:
```n
```python
>>> for i in range(6):
...   results = model.fit(data['Y'])
...   print "AR coefficients:", model.rho
...   rho, sigma = yule_walker(data['Y'] - results.predicted,
...                         order=2,
...                         df=model.df_resid)
...   model = ARModel(model.design, rho)
... AR coefficients: [ 0.  0.]
AR coefficients: [-0.61530877 -1.01542645]
AR coefficients: [-0.72660832 -1.06201457]
AR coefficients: [-0.7220361 -1.05365352]
AR coefficients: [-0.72229201 -1.05408193]
AR coefficients: [-0.722278 -1.05405838]
>>> results.theta
array([ 1.59564228, -0.58562172])
>>> results.t()
array([ 38.0890515 , -3.45429252])
>>> print results.Tcontrast([0, 1])
<T contrast: effect=-0.58562172384377043, sd=0.16953449108110835,
t=-3.4542925165805847, df_den=5>
```

60.2. Classes 463
```python
>>> print results.Fcontrast(np.identity(2))
<F contrast: F=4216.810299725842, df_den=5, df_num=2>

Reinitialize the model, and do the automated iterative fit

```
A dict with key ‘sigma’, which is an estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as \( \text{sum}((Y - X\beta)^2) / n \) where \( n=Y.shape[0] \), \( X=self.design \).

**Returns info**: array

The information matrix, the negative of the inverse of the Hessian of the log-likelihood function evaluated at (theta, Y, nuisance).

**initialize**(design)

**iterative_fit**(Y, niter=3)

Perform an iterative two-stage procedure to estimate AR(p) parameters and regression coefficients simultaneously.

**Parameters**

- **Y** : ndarray
da data to which to fit model
- **niter** : optional, int
  the number of iterations (default 3)

**Returns**

None:

**logL**(beta, Y, nuisance=None)

Returns the value of the loglikelihood function at beta.

Given the whitened design matrix, the loglikelihood is evaluated at the parameter vector, beta, for the dependent variable, Y and the nuisance parameter, sigma.

**Parameters**

- **beta** : ndarray
  The parameter estimates. Must be of length df_model.
- **Y** : ndarray
  The dependent variable
- **nuisance** : dict, optional
  A dict with key ‘sigma’, which is an optional estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as \( \text{sum}((Y - X\beta)^2) / n \), where \( n=Y.shape[0] \), \( X=self.design \).

**Returns**

loglf : float

The value of the loglikelihood function.

**Notes**

The log-Likelihood Function is defined as

\[
\ell(\beta, \sigma, Y) = -\frac{n}{2} \log(2\pi\sigma^2) - \|Y - X\beta\|^2 / (2\sigma^2)
\]

The parameter \( \sigma \) above is what is sometimes referred to as a nuisance parameter. That is, the likelihood is considered as a function of \( \beta \), but to evaluate it, a value of \( \sigma \) is needed.

If \( \sigma \) is not provided, then its maximum likelihood estimate:

\[
\hat{\sigma}(\beta) = \frac{\text{SSE}(\beta)}{n}
\]

is plugged in. This likelihood is now a function of only \( \beta \) and is technically referred to as a profile-likelihood.
References

[R3]

**predict** *(design=None)*

After a model has been fit, results are (assumed to be) stored in self.results, which itself should have a predict method.

**static rank ()**

Compute rank of design matrix

**score (beta, Y, nuisance=None)**

Gradient of the loglikelihood function at (beta, Y, nuisance).

The gradient of the loglikelihood function at (beta, Y, nuisance) is the score function.

See **logL()** for details.

**Parameters**

- **beta**: ndarray
  
The parameter estimates. Must be of length df_model.

- **Y**: ndarray
  
The dependent variable.

- **nuisance**: dict, optional
  
  A dict with key 'sigma', which is an optional estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as \( \text{sum}((Y - X*beta)^2) / n \), where \( n=Y .\text{shape}[0], \ X=self.design \).

**Returns**

The gradient of the loglikelihood function.

**whiten (X)**

Whiten a series of columns according to AR(p) covariance structure

**Parameters**

- **X**: array-like of shape (n_features)
  
  array to whiten

**Returns**

**wX**: ndarray

X whitened with order self.order AR

**60.2.3 GLSModel**

**class** *nipy.algorithms.statistics.models.regression.GLSMModel (design, sigma)*

**Bases**: *nipy.algorithms.statistics.models.regression.OLSModel*

Generalized least squares model with a general covariance structure

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>fit(Y)</strong></td>
<td>Fit model to data Y</td>
</tr>
<tr>
<td><strong>has_intercept()</strong></td>
<td>Check if column of 1s is in column space of design</td>
</tr>
<tr>
<td><strong>information(beta[, nuisance])</strong></td>
<td>Returns the information matrix at (beta, Y, nuisance).</td>
</tr>
<tr>
<td><strong>initialize(design)</strong></td>
<td></td>
</tr>
<tr>
<td><strong>logL(beta, Y[, nuisance])</strong></td>
<td>Returns the value of the loglikelihood function at beta.</td>
</tr>
</tbody>
</table>

Continued on next page
Table 60.3 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>predict([design])</code></td>
<td>After a model has been fit, results are (assumed to be) stored</td>
</tr>
<tr>
<td><code>rank()</code></td>
<td>Compute rank of design matrix</td>
</tr>
<tr>
<td><code>score(beta, Y[, nuisance])</code></td>
<td>Gradient of the loglikelihood function at (beta, Y, nuisance).</td>
</tr>
<tr>
<td><code>whiten(Y)</code></td>
<td></td>
</tr>
</tbody>
</table>

```python
def __init__(design, sigma)
    # Fit model to data Y
    # Full fit of the model including estimate of covariance matrix, (whitened) residuals and scale.

    Parameters
    1. **Y**
       - array-like
       - The dependent variable for the Least Squares problem.

    Returns
    1. **fit**: RegressionResults

    def fit(Y)
        # Fit model to data Y
        # Full fit of the model including estimate of covariance matrix, (whitened) residuals and scale.

    def rank()
        # Compute rank of design matrix

    def score(beta, Y[, nuisance])
        # Gradient of the loglikelihood function at (beta, Y, nuisance).

    def whiten(Y)
        # Whiten the design matrix
```
The value of the loglikelihood function.

**Notes**

The log-Likelihood Function is defined as

\[ \ell(\beta, \sigma, Y) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{||Y - X\beta||^2}{2\sigma^2} \]

The parameter \( \sigma \) above is what is sometimes referred to as a nuisance parameter. That is, the likelihood is considered as a function of \( \beta \), but to evaluate it, a value of \( \sigma \) is needed.

If \( \sigma \) is not provided, then its maximum likelihood estimate:

\[ \hat{\sigma}(\beta) = \frac{\text{SSE}(\beta)}{n} \]

is plugged in. This likelihood is now a function of only \( \beta \) and is technically referred to as a profile-likelihood.

**References**

[R5]

**predict** *(design=None)*

After a model has been fit, results are (assumed to be) stored in self.results, which itself should have a predict method.

**static rank** ()

Compute rank of design matrix

**score** *(beta, Y, nuisance=None)*

Gradient of the loglikelihood function at (beta, Y, nuisance).

The gradient of the loglikelihood function at (beta, Y, nuisance) is the score function.

See **logL()** for details.

**Parameters**

- **beta** : ndarray
  The parameter estimates. Must be of length df_model.

- **Y** : ndarray
  The dependent variable.

- **nuisance** : dict, optional
  A dict with key ‘sigma’, which is an optional estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as \( \text{sum}((Y - X*\text{beta})^2) / n \), where \( n=Y.shape[0], X=\text{self.design} \).

**Returns**

- **The gradient of the loglikelihood function.**

**whiten** *(Y)*
60.2.4 OLSModel

class nipy.algorithms.statistics.models.regression.OLSModel(design)

Bases: nipy.algorithms.statistics.models.model.LikelihoodModel

A simple ordinary least squares model.

Parameters  design : array-like

This is your design matrix. Data are assumed to be column ordered with observations in rows.

Examples

```python
>>> from nipy.algorithms.statistics.api import Term, Formula
>>> data = np.rec.fromarrays(([1,3,4,5,2,3,4], range(1,8)),
...     names=('Y', 'X'))
>>> f = Formula([Term("X"), 1])
>>> dmtx = f.design(data, return_float=True)
>>> model = OLSModel(dmtx)
>>> results = model.fit(data['Y'])
>>> results.theta
array([ 0.25 , 2.14285714])
>>> results.t()
array([ 0.98019606, 1.87867287])
>>> print results.Tcontrast([0,1])
<T contrast: effect=2.14285714286, sd=1.14062281591, t=1.87867287326, df_den=5>
>>> print results.Fcontrast(np.eye(2))
<F contrast: F=19.4607843137, df_den=5, df_num=2>
```

Attributes

Methods

```python
model.__init__(design)
model.logL(b=self.beta, Y)
__init__(design)

Parameters  design : array-like

This is your design matrix. Data are assumed to be column ordered with observations in rows.

fit(Y)

Fit model to data Y

Parameters  Y : array-like

The dependent variable for the Least Squares problem.

Returns  fit : RegressionResults

static has_intercept()

Check if column of 1s is in column space of design
```
**information** *(beta, nuisance=None)*

Returns the information matrix at (beta, Y, nuisance).

See *logL* for details.

**Parameters**

- **beta**: ndarray
  - The parameter estimates. Must be of length df_model.

- **nuisance**: dict
  - A dict with key 'sigma', which is an estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as `\sum((Y - X*beta)**2) / n` where `n=Y.shape[0]`, `X=self.design`.

**Returns**

- **info**: array
  - The information matrix, the negative of the inverse of the Hessian of the log-likelihood function evaluated at (theta, Y, nuisance).

**initialize** *(design)*

**logL** *(beta, Y, nuisance=None)*

Returns the value of the loglikelihood function at beta.

Given the whitened design matrix, the loglikelihood is evaluated at the parameter vector, beta, for the dependent variable, Y and the nuisance parameter, sigma.

**Parameters**

- **beta**: ndarray
  - The parameter estimates. Must be of length df_model.

- **Y**: ndarray
  - The dependent variable

- **nuisance**: dict, optional
  - A dict with key 'sigma', which is an optional estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as `\sum((Y - X*beta)**2) / n`, where `n=Y.shape[0]`, `X=self.design`.

**Returns**

- **loglf**: float
  - The value of the loglikelihood function.

**Notes**

The log-Likelihood Function is defined as

\[
\ell(\beta, \sigma, Y) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{||Y - X\beta||^2}{(2\sigma^2)}
\]

The parameter \(\sigma\) above is what is sometimes referred to as a nuisance parameter. That is, the likelihood is considered as a function of \(\beta\), but to evaluate it, a value of \(\sigma\) is needed.

If \(\sigma\) is not provided, then its maximum likelihood estimate:

\[
\hat{\sigma}(\beta) = \frac{SSE(\beta)}{n}
\]

is plugged in. This likelihood is now a function of only \(\beta\) and is technically referred to as a profile-likelihood.
References

[R7]

```python
predict(design=None)
```

After a model has been fit, results are (assumed to be) stored in self.results, which itself should have a predict method.

```python
static rank()
```

Compute rank of design matrix

```python
score(beta, Y, nuisance=None)
```

Gradient of the loglikelihood function at (beta, Y, nuisance).

The gradient of the loglikelihood function at (beta, Y, nuisance) is the score function.

See `logL()` for details.

**Parameters**

- `beta`: ndarray
  - The parameter estimates. Must be of length df_model.
- `Y`: ndarray
  - The dependent variable.
- `nuisance`: dict, optional
  - A dict with key 'sigma', which is an optional estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as \( \text{sum}((Y - X*beta)^2) / n \), where \( n=Y.shape[0], X=self.design \).

**Returns**

- The gradient of the loglikelihood function.

```python
whiten(X)
```

Whiten design matrix

**Parameters**

- `X`: array
  - design matrix

**Returns**

- `wX`: array
  - This matrix is the matrix whose pseudoinverse is ultimately used in estimating the coefficients. For OLSModel, it does nothing. For WLSmodel, ARmodel, it pre-applies a square root of the covariance matrix to X.

### 60.2.5 RegressionResults

```python
class nipy.algorithms.statistics.models.regression.RegressionResults( theta, Y, model, wY, wresid, cov=None, dispersion=1.0, nuisance=None)
```

**Bases:** `nipy.algorithms.statistics.models.model.LikelihoodModelResults`

This class summarizes the fit of a linear regression model.

It handles the output of contrasts, estimates of covariance, etc.
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AIC()</strong></td>
<td>Akaike Information Criterion</td>
</tr>
<tr>
<td><strong>BIC()</strong></td>
<td>Schwarz’s Bayesian Information Criterion</td>
</tr>
<tr>
<td><strong>F_overall()</strong></td>
<td>Overall goodness of fit F test.</td>
</tr>
<tr>
<td><strong>Fcontrast()</strong></td>
<td>Compute an Fcontrast for a contrast matrix matrix.</td>
</tr>
<tr>
<td><strong>MSE()</strong></td>
<td>Mean square (error)</td>
</tr>
<tr>
<td><strong>MSR()</strong></td>
<td>Mean square (regression)</td>
</tr>
<tr>
<td><strong>MST()</strong></td>
<td>Mean square (total)</td>
</tr>
<tr>
<td><strong>R2()</strong></td>
<td>Return the adjusted R^2 value for each row of the response Y.</td>
</tr>
<tr>
<td><strong>R2_adj()</strong></td>
<td>Return the R^2 value for each row of the response Y.</td>
</tr>
<tr>
<td><strong>SSE()</strong></td>
<td>Error sum of squares.</td>
</tr>
<tr>
<td><strong>SSR()</strong></td>
<td>Regression sum of squares</td>
</tr>
<tr>
<td><strong>SST()</strong></td>
<td>Total sum of squares.</td>
</tr>
<tr>
<td><strong>Tcontrast()</strong></td>
<td>Compute a Tcontrast for a row vector matrix</td>
</tr>
<tr>
<td><strong>conf_int()</strong></td>
<td>The confidence interval of the specified theta estimates.</td>
</tr>
<tr>
<td><strong>logL()</strong></td>
<td>The maximized log-likelihood</td>
</tr>
<tr>
<td><strong>norm_resid()</strong></td>
<td>Residuals, normalized to have unit length.</td>
</tr>
<tr>
<td><strong>predicted()</strong></td>
<td>Return linear predictor values from a design matrix.</td>
</tr>
<tr>
<td><strong>resid()</strong></td>
<td>Residuals from the fit.</td>
</tr>
<tr>
<td><strong>t()</strong></td>
<td>Return the (Wald) t-statistic for a given parameter estimate.</td>
</tr>
<tr>
<td><strong>vcov()</strong></td>
<td>Variance/covariance matrix of linear contrast</td>
</tr>
</tbody>
</table>

**_init_(theta, Y, model, wY, wresid, cov=None, dispersion=1.0, nuisance=None)**

See LikelihoodModelResults constructor.

The only difference is that the whitened Y and residual values are stored for a regression model.

**static AIC()**
Akaike Information Criterion

**static BIC()**
Schwarz’s Bayesian Information Criterion

**static F_overall()**
Overall goodness of fit F test, comparing model to a model with just an intercept. If not an OLS model this is a pseudo-F.

**Fcontrast()**
Compute an Fcontrast for a contrast matrix matrix.

Here, matrix M is assumed to be non-singular. More precisely

\[ Mp \hat{X} p' \]

is assumed invertible. Here, \( pX \) is the generalized inverse of the design matrix of the model. There can be problems in non-OLS models where the rank of the covariance of the noise is not full.

See the contrast module to see how to specify contrasts. In particular, the matrices from these contrasts will always be non-singular in the sense above.

**Parameters**
- **matrix**: 1D array-like
  - contrast matrix
- **dispersion**: None or float, optional
  - If None, use self.dispersion

Chapter 60. algorithms.statistics.models.regression
invcov : None or array, optional
    Known inverse of variance covariance matrix. If None, calculate this matrix.

Returns f_res : FContrastResults instance
    with attributes F, df_den, df_num

Notes

For F contrasts, we now specify an effect and covariance

static MSE ()
    Mean square (error)

static MSR ()
    Mean square (regression)

static MST ()
    Mean square (total)

static R2 ()
    Return the adjusted $R^2$ value for each row of the response $Y$.

Notes

Changed to the textbook definition of $R^2$.
See: Davidson and MacKinnon p 74

static R2_adj ()
    Return the $R^2$ value for each row of the response $Y$.

Notes

Changed to the textbook definition of $R^2$.
See: Davidson and MacKinnon p 74

static SSE ()
    Error sum of squares. If not from an OLS model this is “pseudo”-SSE.

static SSR ()
    Regression sum of squares

static SST ()
    Total sum of squares. If not from an OLS model this is “pseudo”-SST.

Tcontrast(matrix, store=('t', 'effect', 'sd'), dispersion=None)
    Compute a Tcontrast for a row vector matrix

To get the t-statistic for a single column, use the ‘t’ method.

Parameters matrix : 1D array-like
    contrast matrix

store : sequence, optional
    components of t to store in results output object. Defaults to all components ('t', 'effect', 'sd').
dispersion : None or float, optional

Returns res : TContrastResults object

cnf_int (alpha=0.05, cols=None, dispersion=None)
The confidence interval of the specified theta estimates.

Parameters alpha : float, optional

The alpha level for the confidence interval. ie., alpha = .05 returns a 95% confidence interval.

cols : tuple, optional

cols specifies which confidence intervals to return

dispersion : None or scalar

scale factor for the variance / covariance (see class docstring and vcov method docstring)

Returns cis : ndarray

cis is shape (len(cols), 2) where each row contains [lower, upper] for the given entry in cols

Notes

Confidence intervals are two-tailed. TODO: tails : string, optional

tails can be “two”, “upper”, or “lower”

Examples

```python
>>> from numpy.random import standard_normal as stan
>>> from nipy.algorithms.statistics.models.regression import OLSModel
>>> x = np.hstack((stan((30,1)),stan((30,1)),stan((30,1))))
>>> beta=np.array([3.25, 1.5, 7.0])
>>> y = np.dot(x,beta) + stan((30))
>>> model = OLSModel(x).fit(y)
>>> confidence_intervals = model.conf_int(cols=(1,2))
```

static logL ()
The maximized log-likelihood

static norm_resid ()
Residuals, normalized to have unit length.

Notes

Is this supposed to return “stanardized residuals,” residuals standardized to have mean zero and approximately unit variance?

d_i = e_i / sqrt(MS_E)

Where MS_E = SSE / (n - k)

See: Montgomery and Peck 3.2.1 p. 68  Davidson and MacKinnon 15.2 p 662
static predicted()
    Return linear predictor values from a design matrix.

static resid()
    Residuals from the fit.

t (column=None)
    Return the (Wald) t-statistic for a given parameter estimate.

    Use Tcontrast for more complicated (Wald) t-statistics.

vcov (matrix=None, column=None, dispersion=None, other=None)
    Variance/covariance matrix of linear contrast

    Parameters
    matrix: (dim, self.theta.shape[0]) array, optional
        numerical contrast specification, where dim refers to the ‘dimension’ of the contrast i.e.
        1 for t contrasts, 1 or more for F contrasts.

    column: int, optional
        alternative way of specifying contrasts (column index)

    dispersion: float or (n_voxels,) array, optional
        value(s) for the dispersion parameters

    other: (dim, self.theta.shape[0]) array, optional
        alternative contrast specification (?)

    Returns
    cov: (dim, dim) or (n_voxels, dim, dim) array
        the estimated covariance matrix/matrices

    Returns the variance/covariance matrix of a linear contrast of the:
    estimates of theta, multiplied by ‘dispersion’ which will often be an:
    estimate of ‘dispersion’, like, sigma^2.

    The covariance of interest is either specified as a (set of) column(s):
    or a matrix.

60.2.6  WLSModel

class nipy.algorithms.statistics.models.regression.WLSModel (design, weights=1)
    Bases: nipy.algorithms.statistics.models.regression.OLSModel

    A regression model with diagonal but non-identity covariance structure.

    The weights are presumed to be (proportional to the) inverse of the variance of the observations.

Examples

    >>> from nipy.algorithms.statistics.api import Term, Formula
    >>> data = np.rec.fromarrays([(1,3,4,5,2,3,4), range(1,8)],
    ...               names=('Y', 'X'))
    >>> f = Formula([Term("X"), 1])
    >>> dmtx = f.design(data, return_float=True)
    >>> model = WLSModel(dmtx, weights=range(1,8))
    >>> results = model.fit(data['Y'])
```python
>>> results.theta
array([[ 0.0952381 , 2.91666667]])
>>> results.t()
array([[ 0.35684428, 2.0652652 ]])
>>> print results.Tcontrast([0,1])
<T contrast: effect=2.91666666667, sd=1.41224801095, t=2.06526519708, df_den=5>
>>> print results.Fcontrast(np.identity(2))
<F contrast: F=26.9986072423, df_den=5, df_num=2>
```

## Methods

- **fit(Y)**: Fit model to data $Y$
- **has_intercept()**: Check if column of 1s is in column space of design.
- **information(beta[, nuisance])**: Returns the information matrix at (beta, $Y$, nuisance).
- **initialize(design)**
- **logL(beta, Y[, nuisance])**: Returns the value of the loglikelihood function at beta.
- **predict([design])**: After a model has been fit, results are (assumed to be) stored
- **rank()**: Compute rank of design matrix
- **score(beta, Y[, nuisance])**: Gradient of the loglikelihood function at (beta, $Y$, nuisance).
- **whiten(X)**: Whitener for WLS model, multiplies by sqrt(self.weights)

```python
__init__(design, weights=1)

fit(Y)
Fit model to data $Y$

Full fit of the model including estimate of covariance matrix, (whitened) residuals and scale.

Parameters

- **Y**: array-like
  The dependent variable for the Least Squares problem.

Returns

- **fit**: RegressionResults

static has_intercept()
Check if column of 1s is in column space of design

information (beta, nuisance=\text{None})
Returns the information matrix at (beta, $Y$, nuisance).

See logL for details.

Parameters

- **beta**: ndarray
  The parameter estimates. Must be of length df_model.

- **nuisance**: dict
  A dict with key ‘sigma’, which is an estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as $\sum((Y - X*beta)^2) / n$ where $n=Y.shape[0]$, $X=\text{self.design}$.

Returns

- **info**: array
  The information matrix, the negative of the inverse of the Hessian of the loglikelihood function evaluated at (theta, $Y$, nuisance).

initialize(design)
**logL** *(beta, Y, nuisance=None)*

Returns the value of the loglikelihood function at beta.

Given the whitened design matrix, the loglikelihood is evaluated at the parameter vector, beta, for the dependent variable, Y and the nuisance parameter, sigma.

**Parameters**

- **beta**: ndarray
  The parameter estimates. Must be of length df_model.

- **Y**: ndarray
  The dependent variable

- **nuisance**: dict, optional
  A dict with key ‘sigma’, which is an optional estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as \( \text{sum}((Y - X*beta)^2) / n \), where \( n=Y \text{.shape[0]} \), \( X=\text{self.design} \).

**Returns**

- **loglf**: float
  The value of the loglikelihood function.

**Notes**

The log-Likelihood Function is defined as

\[
\ell(\beta, \sigma, Y) = -\frac{n}{2} \log(2\pi\sigma^2) - \frac{||Y - X\beta||^2}{2\sigma^2}
\]

The parameter \( \sigma \) above is what is sometimes referred to as a nuisance parameter. That is, the likelihood is considered as a function of \( \beta \), but to evaluate it, a value of \( \sigma \) is needed.

If \( \sigma \) is not provided, then its maximum likelihood estimate:

\[
\hat{\sigma}(\beta) = \frac{\text{SSE}(\beta)}{n}
\]

is plugged in. This likelihood is now a function of only \( \beta \) and is technically referred to as a profile-likelihood.

**References**

[R8]

**predict** *(design=None)*

After a model has been fit, results are (assumed to be) stored in self.results, which itself should have a predict method.

**static rank ()**

Compute rank of design matrix

**score** *(beta, Y, nuisance=None)*

Gradient of the loglikelihood function at (beta, Y, nuisance).

The gradient of the loglikelihood function at (beta, Y, nuisance) is the score function.

See **logL()** for details.

**Parameters**

- **beta**: ndarray
  The parameter estimates. Must be of length df_model.
The dependent variable.

**nuisance** : dict, optional
A dict with key 'sigma', which is an optional estimate of sigma. If None, defaults to its maximum likelihood estimate (with beta fixed) as \[ \text{sum}((Y - X*beta)^2) / n, \]
where \( n=Y\text{.shape}[0], X=self\text{.design}. \)

**Returns**
The gradient of the loglikelihood function.

**whiten** \((X)\)
Whitener for WLS model, multiplies by \( \sqrt{\text{self.weights}} \)

### 60.3 Functions

**nipy.algorithms.statistics.models.regression.ar_bias_correct**

\[
\text{ar_bias_correct}(\text{results, order, invM=None})
\]

Apply bias correction in calculating AR(p) coefficients from \text{results}

There is a slight bias in the rho estimates on residuals due to the correlations induced in the residuals by fitting a linear model. See [Worsley2002].

This routine implements the bias correction described in appendix A.1 of [Worsley2002].

**Parameters**

- **results** : ndarray or results object
  If ndarray, assume these are residuals, from a simple model. If a results object, with attribute \text{resid}, then use these for the residuals. See Notes for more detail

- **order** : int
  Order p of AR(p) model

- **invM** : None or array
  Known bias correcting matrix for covariance. If None, calculate from \text{results.model}

**Returns**

- **rho** : array
  Bias-corrected AR(p) coefficients

**Notes**

If \text{results} has attributes \text{resid} and \text{scale}, then assume \text{scale} has come from a fit of a potentially customized model, and we use that for the sum of squared residuals. In this case we also need \text{results.df_resid}. Otherwise we assume this is a simple Gaussian model, like OLS, and take the simple sum of squares of the residuals.

**References**

[Worsley2002]

**nipy.algorithms.statistics.models.regression.ar_bias_corrector**

\[
\text{ar_bias_corrector}(\text{design, calc_beta, order=1})
\]

Return bias correcting matrix for \text{design} and AR order \text{order}
There is a slight bias in the rho estimates on residuals due to the correlations induced in the residuals by fitting a linear model. See [Worsley2002].

This routine implements the bias correction described in appendix A.1 of [Worsley2002].

**Parameters**  
*design*: array  
Design matrix

*calc_beta*: array  
Moore-Penrose pseudoinverse of the (maybe) whitened design matrix. This is the matrix that, when applied to the (maybe whitened) data, produces the betas.

*order*: int, optional  
Order p of AR(p) process

**Returns**  
invM*: array  
Matrix to bias correct estimated covariance matrix in calculating the AR coefficients

**References**

[Worsley2002]

```python
def isestimable(C, D)
    # checks if the contrast C is estimable by looking at the rank of vstack([C,D])
    return np.linalg.matrix_rank(np.vstack([C,D])) == np.linalg.matrix_rank(D)
```

**Parameters**  
*C*: (Q, P) array-like  
contrast matrix. If C has is 1 dimensional assume shape (1, P)

*D*: (N, P) array-like  
design matrix

**Returns**  
*tf*: bool  
True if the contrast C is estimable on design D

**Examples**

```python
>>> D = np.array([(1, 1, 1, 0, 0, 0),
                (0, 0, 1, 1, 1),
                (1, 1, 1, 1, 1)]).T
>>> isestimable([1, 0, 0], D)
False
>>> isestimable([1, -1, 0], D)
True
```

```python
def yule_walker(X, order=1, method='unbiased', df=None, inv=False)
```

Estimate AR(p) parameters from a sequence X using Yule-Walker equation.

unbiased or maximum-likelihood estimator (mle)

See, for example:
Parameters

- **X**: ndarray of shape(n)
  - **order**: int, optional
    - Order of AR process.
  - **method**: str, optional
    - Method can be “unbiased” or “mle” and this determines denominator in estimate of autocorrelation function (ACF) at lag k. If “mle”, the denominator is n=X.shape[0], if “unbiased” the denominator is n-k.
  - **df**: int, optional
    - Specifies the degrees of freedom. If df is supplied, then it is assumed the X has df degrees of freedom rather than n.
  - **inv**: bool, optional
    - Whether to return the inverse of the R matrix (see code)

Returns

- **rho**: (order,) ndarray
  - **sigma**: int
    - standard deviation of the residuals after fit
  - **R_inv**: ndarray
    - If inv is True, also return the inverse of the R matrix

Notes

See also http://en.wikipedia.org/wiki/AR_model#Calculation_of_the_AR_parameters
61.1 Module: algorithms.statistics.models.setupscons

nipy.algorithms.statistics.models.setupscons.configuration(parent_package=''
, top_path=\text{None})
ALGORITHMS STATISTICS MODELS UTILS

62.1 Module: algorithms.statistics.models.utils

Inheritance diagram for nipy.algorithms.statistics.models.utils:

```
models.utils.StepFunction
```

General matrix and other utilities for statistics

62.2 Class

62.3 StepFunction

```
class nipy.algorithms.statistics.models.utils.StepFunction(x, y, ival=0.0, sorted=False):
    Bases: object
    A basic step function: values at the ends are handled in the simplest way possible: everything to the left of x[0] is set to ival; everything to the right of x[-1] is set to y[-1].
```

Examples

```
>>> x = np.arange(20)
>>> y = np.arange(20)
>>> f = StepFunction(x, y)

>>> print f(3.2)
3.0
>>> print f([[3.2,4.5],[24,-3.1]])
[[  3.  4.]
 [19.  0.]]
```
Methods

__call__(time)

__init__(x, y, ival=0.0, sorted=False)

62.4 Functions

nipy.algorithms.statistics.models.utils.ECDF(values)
   Return the ECDF of an array as a step function.

nipy.algorithms.statistics.models.utils.mad(a, c=0.6745, axis=0)
   Median Absolute Deviation:
   median(abs(a - median(a))) / c

nipy.algorithms.statistics.models.utils.monotone_fn_inverter(fn, x, vectorized=True, **keywords)
   Given a monotone function x (no checking is done to verify monotonicity) and a set of x values, return an linearly interpolated approximation to its inverse from its values on x.
ALGORITHMS.STATISTICS.ONESAMPLE

63.1 Module: algorithms.statistics.onesample

Utilities for one sample t-tests

63.2 Functions

nipy.algorithms.statistics.onesample.estimate_mean(Y, sd)
Estimate the mean of a sample given information about the standard deviations of each entry.

Parameters  
Y : ndarray
Data for which mean is to be estimated. Should have shape[0] == number of subjects.

sd : ndarray
Standard deviation (subject specific) of the data for which the mean is to be estimated.
Should have shape[0] == number of subjects.

Returns  
value : dict
This dictionary has keys ['effect', 'scale', 't', 'resid', 'sd']

nipy.algorithms.statistics.onesample.estimate_varatio(Y, sd, df=None, niter=10)
Estimate variance fixed/random effects variance ratio

In a one-sample random effects problem, estimate the ratio between the fixed effects variance and the random effects variance.

Parameters  
Y : np.ndarray
Data for which mean is to be estimated. Should have shape[0] == number of subjects.

sd : array
Standard deviation (subject specific) of the data for which the mean is to be estimated.
Should have shape[0] == number of subjects.

df : int or None, optional
If supplied, these are used as weights when deriving the fixed effects variance. Should have length == number of subjects.

niter : int, optional
Number of EM iterations to perform (default 10)
Returns  value : dict

This dictionary has keys ['fixed', 'ratio', 'random'], where 'fixed' is the fixed effects variance implied by the input parameter 'sd'; 'random' is the random effects variance and 'ratio' is the estimated ratio of variances: 'random'/'fixed'.
### 64.1 Module: algorithms.statistics.rft

Inheritance diagram for `nipy.algorithms.statistics.rft`:

Random field theory routines

The theoretical results for the EC densities appearing in this module were partially supported by NSF grant DMS-0405970.


### 64.2 Classes

#### 64.2.1 ChiBarSquared

```python
class nipy.algorithms.statistics.rft.ChiBarSquared (dfn=1, search=[1])
Bases: nipy.algorithms.statistics.rft.ChiSquared
```
Methods

__call__(x[, dim, search])
density(x, dim) The EC density in dimension dim.
integ([m, k])
pvalue(x[, search])
quasi(dim) (Quasi-)polynomial parts of EC density in dimension dim

__init__(dfn=1, search=[1])
density(x, dim)
   The EC density in dimension dim.
integ(m=None, k=None)
pvalue(x, search=None)
quasi(dim)
   (Quasi-)polynomial parts of EC density in dimension dim
   • ignoring a factor of (2pi)^{-(dim+1)/2} in front.

64.2.2 ChiSquared

class nipy.algorithms.statistics.rft.ChiSquared(dfn, dfd=inf, search=[1])
   Bases: nipy.algorithms.statistics.rft.ECcone

   EC densities for a Chi-Squared(n) random field.

Methods

__call__(x, search])
density(x, dim) The EC density in dimension dim.
integ([m, k])
pvalue(x[, search])
quasi(dim) (Quasi-)polynomial parts of EC density in dimension dim

__init__(dFn, dfd=inf, search=[1])
density(x, dim)
   The EC density in dimension dim.
integ(m=None, k=None)
pvalue(x, search=None)
quasi(dim)
   (Quasi-)polynomial parts of EC density in dimension dim
   • ignoring a factor of (2pi)^{-(dim+1)/2} in front.
64.2.3 ECcone

class nipy.algorithms.statistics.rft.ECcone(mu=[1], dfd=inf, search=[1], product=[1])
Bases: nipy.algorithms.statistics.rft.IntrinsicVolumes

EC approximation to supremum distribution of var==1 Gaussian process

A class that takes the intrinsic volumes of a set and gives the EC approximation to the supremum distribution of a unit variance Gaussian process with these intrinsic volumes. This is the basic building block of all of the EC densities.

If product is not None, then this product (an instance of IntrinsicVolumes) will effectively be prepended to the search region in any call, but it will also affect the (quasi-)polynomial part of the EC density. For instance, Hotelling’s $T^2$ random field has a sphere as product, as does Roy’s maximum root.

**Methods**

__call__ (x[, search]) Get expected EC for a search region

density(x, dim) The EC density in dimension dim.

integ([m, k])
pvalue(x[, search])
quasi(dim) (Quasi-)polynomial parts of EC density in dimension dim

__init__ (mu=[1], dfd=inf, search=[1], product=[1])

density (x, dim)

  The EC density in dimension dim.

integ (m=None, k=None)
pvalue (x, search=None)
quasi (dim)

  (Quasi-)polynomial parts of EC density in dimension dim

  • ignoring a factor of $(2\pi)^{-(dim+1)/2}$ in front.

64.2.4 ECquasi

class nipy.algorithms.statistics.rft.ECquasi(c_or_r, r=0, exponent=None, m=None)
Bases: numpy.lib.polynomial.poly1d

Polynomials with premultiplier

A subclass of poly1d consisting of polynomials with a premultiplier of the form:

$$(1 + x^2/m)^{-\text{exponent}}$$

where $m$ is a non-negative float (possibly infinity, in which case the function is a polynomial) and exponent is a non-negative multiple of 1/2.

These arise often in the EC densities.
Examples

```python
>>> import numpy
>>> from nipy.algorithms.statistics.rft import ECquasi

>>> x = numpy.linspace(0,1,101)

>>> a = ECquasi([3,4,5])
>>> a
ECquasi(array([3, 4, 5]), m=inf, exponent=0.000000)

>>> a(3) == 3*3**2 + 4*3 + 5
True

>>> b = ECquasi(a.coeffs, m=30, exponent=4)
>>> numpy.allclose(b(x), a(x) * numpy.power(1+x**2/30, -4))
True
```

Attributes

- `coeffs`
- `order`
- `variable`

Methods

- `__call__`(val) Evaluate the ECquasi instance.
- `change_exponent`(_pow) Change exponent
- `compatible`(other) Check compatibility of degrees of freedom
- `denom_poly`() Base of the premultiplier: \((1+x^2/m)\).
- `deriv`([m]) Evaluate derivative of ECquasi
- `integ`([m, k]) Return an antiderivative (indefinite integral) of this polynomial.

```
>>> import numpy

>>> b = ECquasi([3,4,20], m=30, exponent=4)

>>> x = numpy.linspace(0,1,101)

>>> c = b.change_exponent(3)

>>> numpy.allclose(c(x), b(x))
True
```

Examples

```python
>>> b = ECquasi([3,4,20], m=30, exponent=4)

>>> x = numpy.linspace(0,1,101)

>>> c = b.change_exponent(3)

>>> numpy.allclose(c(x), b(x))
True
```
```python
coeffs = None

compatible(other)
    Check compatibility of degrees of freedom
    Check whether the degrees of freedom of two instances are equal so that they can be multiplied together.

Examples

>>> import numpy

>>> b = ECquasi([3,4,20], m=30, exponent=4)
>>> x = numpy.linspace(0,1,101)
>>> c = b.change_exponent(3)
>>> b.compatible(c)
True

>>> d = ECquasi([3,4,20])
True

 denom_poly()
    Base of the premultiplier: (1+x^2/m).

Examples

>>> import numpy

>>> b = ECquasi([3,4,20], m=30, exponent=4)
>>> d = b.denom_poly()
>>> d
poly1d([ 0.03333333, 0. , 1. ])

>>> numpy.allclose(d.c, [1./b.m,0,1])
True

deriv(m=1)
    Evaluate derivative of ECquasi

    Parameters  m : int, optional

Examples

>>> a = ECquasi([3,4,5])
>>> a.deriv(m=2)
ECquasi(array([6]), m=inf, exponent=0.000000)

>>> b = ECquasi([3,4,5], m=10, exponent=3)
>>> b.deriv()
ECquasi(array([-1.2, -2. ,  3. ,  4. ]), m=10.000000, exponent=4.000000)

integ(m=1, k=0)
    Return an antiderivative (indefinite integral) of this polynomial.
    Refer to polyint for full documentation.

See Also:

polyint equivalent function
```

64.2. Classes
order = None
variable = None

64.2.5 FStat

class nipy.algorithms.statistics.rft.FStat (dfn, dfd=inf, search=[1])
Bases: nipy.algorithms.statistics.rft.ECcone

EC densities for a F random field.

Methods

__call__([x, search])
density(x, dim) The EC density in dimension dim.
integ([m, k])
pvalue([x, search])
quasi(dim) (Quasi-)polynomial parts of EC density in dimension dim

__init__(dfn, dfd=inf, search=[1])
density(x, dim)
The EC density in dimension dim.
intrag(m=none, k=none)
pvalue(x, search=none)
quasi(dim) (Quasi-)polynomial parts of EC density in dimension dim

492 Chapter 64. algorithms.statistics.rft
**density** *(x, dim)*

The EC density in dimension *dim*.

**integ** *(m=None, k=None)*

**pvalue** *(x, search=None)*

**quasi** *(dim)*

(Quasi-)polynomial parts of EC density in dimension *dim*

• ignoring a factor of \((2\pi)^{-\frac{\left\lfloor (\text{dim}+1)/2 \right\rfloor}{\text{dim}}}\) in front.

### 64.2.7 IntrinsicVolumes

**class** *nipy.algorithms.statistics.rft.IntrinsicVolumes*(*mu=[1]*)

* Bases: object
  * Compute intrinsic volumes of products of sets
  * A simple class that exists only to compute the intrinsic volumes of products of sets (that themselves have intrinsic volumes, of course).

**__init__** *(mu=[1])*

### 64.2.8 MultilinearForm

**class** *nipy.algorithms.statistics.rft.MultilinearForm*(*dims, **keywords*)

* Bases: *nipy.algorithms.statistics.rft.ECcone*
  * Maximize a multivariate Gaussian form
  * Maximized over spheres of dimension *dims*. See:

**Methods**

**__call__** *(x[, search]) Get expected EC for a search region*

**density** *(x, dim)* The EC density in dimension *dim*.

**integ** *(m, k)*

**pvalue** *(x[, search])* 

**quasi** *(dim)* (Quasi-)polynomial parts of EC density in dimension *dim*

• ignoring a factor of \((2\pi)^{-\frac{(\text{dim}+1)/2}{\text{dim}}}\) in front.
64.2.9 OneSidedF

class nipy.algorithms.statistics.rft.OneSidedF (dfn, dfd=inf, search=[1])

Bases: nipy.algorithms.statistics.rft.ECcone

EC densities for one-sided F statistic

See:

Methods

__call__(x[, search])

density(x, dim) The EC density in dimension dim.

integ([m, k])
pvalue(x[, search])
quasi(dim) (Quasi-)polynomial parts of EC density in dimension dim

__init__(dfn, dfd=inf, search=[1])

density (x, dim)
The EC density in dimension dim.

integ (m=None, k=None)
pvalue (x, search=None)
quasi (dim)
(Quasi-)polynomial parts of EC density in dimension dim

• ignoring a factor of (2pi)^{-(dim+1)/2} in front.

64.2.10 Roy

class nipy.algorithms.statistics.rft.Roy (dfn=1, dfd=inf, k=1, search=[1])

Bases: nipy.algorithms.statistics.rft.ECcone

Roy’s maximum root

Maximize an F_{dfd,dfn} statistic over a sphere of dimension k.

Methods

__call__(x[, search])

density(x, dim) The EC density in dimension dim.

integ([m, k])
pvalue(x[, search])
quasi(dim) (Quasi-)polynomial parts of EC density in dimension dim

__init__(dfn=1, dfd=inf, k=1, search=[1])

density (x, dim)
The EC density in dimension \( \text{dim} \).

\[ \text{integ}(m=\text{None}, k=\text{None}) \]
\[ \text{pvalue}(x, \text{search}=\text{None}) \]
\[ \text{quasi}(\text{dim}) \]

(Quasi-)polynomial parts of EC density in dimension \( \text{dim} \)

\[ \text{• ignoring a factor of } (2\pi)^{-\left(\text{dim}+1\right)/2} \text{ in front.} \]

### 64.2.11 TStat

**class** nipy.algorithms.statistics.rft.TStat (\( dfd=\text{inf}, \text{search}=[1] \))

Bases: nipy.algorithms.statistics.rft.ECcone

EC densities for a t random field.

**Methods**

\[ \text{__call__}(x[, \text{search}]) \]

Get expected EC for a search region

\[ \text{density}(x, \text{dim}) \]

The EC density in dimension \( \text{dim} \).

\[ \text{integ}([m, k]) \]

\[ \text{pvalue}(x[, \text{search}]) \]

\[ \text{quasi}(\text{dim}) \]

(Quasi-)polynomial parts of EC density in dimension \( \text{dim} \)

\[ \text{• ignoring a factor of } (2\pi)^{-\left(\text{dim}+1\right)/2} \text{ in front.} \]

### 64.2.12 fnsum

**class** nipy.algorithms.statistics.rft.fnsum (*items)

Bases: object

**Methods**

\[ \text{__call__}(x) \]

\[ \text{__init__}(*\text{items}) \]

64.2. Classes 495
64.3 Functions

nipy.algorithms.statistics.rft.Q(dim, d fd=inf)

Q polynomial

If d fd == inf (the default), then Q(dim) is the (dim-1)-st Hermite polynomial:

\[ H_j(x) = (-1)^j e^{x^2/2} \frac{d^j}{dx^j} e^{-x^2/2} \]

If d fd != inf, then it is the polynomial Q defined in [Worsley1994]

Parameters  
\- dim : int  
  dimension of polynomial  
\- d fd : scalar  

Returns  
\- q_poly : np.poly1d instance

References

[Worsley1994]

nipy.algorithms.statistics.rft.ball_search(n, r=1)

A ball-shaped search region of radius r.

nipy.algorithms.statistics.rft.binomial(n, k)

Binomial coefficient

\[ c = \frac{n!}{(n-k)! \, k!} \]

Parameters  
\- n : float  
  n of (n, k)  
\- k : float  
  k of (n, k)

Returns  
\- c : float

Examples

First 3 values of 4 th row of Pascal triangle

```python
>>> [binomial(4, k) for k in range(3)]
[1.0, 4.0, 6.0]
```

nipy.algorithms.statistics.rft.mu_ball(n, j, r=1)

j-th curvature of 'n'-dimensional ball radius r

Return mu_j(B_n(r)), the j-th Lipschitz Killing curvature of the ball of radius r in R^n.

nipy.algorithms.statistics.rft.mu_sphere(n, j, r=1)

j-th curvature for 'n' dimensional sphere radius r

Return mu_j(S_r(R^n)), the j-th Lipschitz Killing curvature of the sphere of radius r in R^n.

From Chapter 6 of

```python
nipy.algorithms.statistics.rft.scale_space(region, interval, kappa=1.0)
```

scale space intrinsic volumes of region x interval

See:


and


```python
nipy.algorithms.statistics.rft.spherical_search(n, r=1)
```

A spherical search region of radius r.

```python
nipy.algorithms.statistics.rft.volume2ball(vol, d=3)
```

Approximate volume with ball

Approximate intrinsic volumes of a set with a given volume by those of a ball with a given dimension and equal volume.
65.1 Module: `algorithms.statistics.utils`

65.2 Functions

`nipy.algorithms.statistics.utils.combinations(iterable, r)`

`nipy.algorithms.statistics.utils.complex(maximal=[(0, 3, 2, 7), (0, 6, 2, 7), (0, 7, 5, 4), (0, 7, 5, 1), (0, 7, 4, 6), (0, 3, 1, 7)])`

Faces from simplices

Take a list of maximal simplices (by default a triangulation of a cube into 6 tetrahedra) and computes all faces

Parameters

- `maximal`: sequence of sequences, optional
  - Default is triangulation of cube into tetrahedra

Returns

- `faces`: dict

`nipy.algorithms.statistics.utils.cube_with_strides_center(center=[0, 0, 0], strides=[4, 2, 1])`

Cube in an array of voxels with a given center and strides.

This triangulates a cube with vertices `center[i] + 1`.

The dimension of the cube is determined by `len(center)` which should agree with `len(center)`. The allowable dimensions are [1,2,3].

Parameters

- `center`: (d,) sequence of int, optional
  - Default is [0, 0, 0]
- `strides`: (d,) sequence of int, optional
  - Default is [4, 2, 1]. These are the strides given by `np.ones((2,2,2), np.bool).strides`

Returns

- `complex`: dict
  - A dictionary with integer keys representing a simplicial complex. The vertices of the simplicial complex are the indices of the corners of the cube in a ‘flattened’ array with specified strides.

`nipy.algorithms.statistics.utils.decompose2d(shape, dim=3)`

Return all (dim-1)-dimensional simplices in a triangulation of a square of a given shape. The vertices in the triangulation are indices in a ‘flattened’ array of the specified shape.
nipy.algorithms.statistics.utils.decompose3d(shape, dim=4)
Return all (dim-1)-dimensional simplices in a triangulation of a cube of a given shape. The vertices in the triangulation are indices in a ‘flattened’ array of the specified shape.

nipy.algorithms.statistics.utils.join_complexes(*complexes)
Join a sequence of simplicial complexes.
Returns the union of all the particular faces.

nipy.algorithms.statistics.utils.multiple_fast_inv(a)
Compute the inverse of a set of arrays.

Parameters
a: array_like of shape (n_samples, n_dim, n_dim):
Set of square matrices to be inverted. A is changed in place.

Returns
a: ndarray:
yielding the inverse of the inputs

Raises LinAlgError:
If a is singular.

ValueError:
If a is not square, or not 2-dimensional.

Notes
This function is borrowed from scipy.linalg.inv, but with some customizations for speed-up.

nipy.algorithms.statistics.utils.multiple_mahalanobis(effect, covariance)
Returns the squared Mahalanobis distance for a given set of samples

Parameters
effect: array of shape (n_features, n_samples),
Each column represents a vector to be evaluated
covariance: array of shape (n_features, n_features, n_samples),
Corresponding covariance models stacked along the last axis

Returns sqd: array of shape (n_samples,)
the squared distances (one per sample)

nipy.algorithms.statistics.utils.test_EC2(shape)
nipy.algorithms.statistics.utils.test_EC3(shape)
nipy.algorithms.statistics.utils.z_score(pvalue)
Return the z-score corresponding to a given p-value.
66.1 Module: \texttt{algorithms.utils.fast\_distance}

this module contains a function to perform fast distance computation on arrays

\texttt{Author}: Bertrand Thirion, 2008-2011

\texttt{nipy.algorithms.utils.fast\_distance.euclidean\_distance}(X, Y=None)

Considering the rows of X (and Y=X) as vectors, compute the distance matrix between each pair of vectors

\textbf{Parameters}
\begin{itemize}
  \item X, array of shape (n1,p) :
  \item Y=None, array of shape (n2,p) :
    \begin{itemize}
      \item if Y=None, then Y=X is used instead
    \end{itemize}
\end{itemize}

\textbf{Returns}
\begin{itemize}
  \item ED, array fo shape(n1, n2) with all the pairwise distance :
\end{itemize}
67.1 Module: `algorithms.utils.matrices`

Utilities for working with matrices

67.2 Functions

`nipy.algorithms.utils.matrices.full_rank(X, r=None)`

Return full-rank matrix whose column span is the same as `X`

Uses an SVD decomposition.

If the rank of `X` is known it can be specified by `r` – no check is made to ensure that this really is the rank of `X`.

**Parameters**

`X` : array-like

2D array which may not be of full rank.

`r` : None or int

Known rank of `X`. `r=None` results in standard matrix rank calculation. We do not check `r` is really the rank of `X`; it is to speed up calculations when the rank is already known.

**Returns**

`fx` : array

Full-rank matrix with column span matching that of `X`

`nipy.algorithms.utils.matrices.matrix_rank(M, tol=None)`

Return rank of matrix using SVD method

Rank of the array is the number of SVD singular values of the array that are greater than `tol`.

This version of matrix rank is very similar to the `numpy.linalg` version except for the use of:

- `scipy.linalg.svd` instead of `numpy.linalg.svd`.
- the MATLAB algorithm for default tolerance calculation

`matrix_rank` appeared in `numpy.linalg` in December 2009, first available in `numpy 1.5.0`.

**Parameters**

`M` : array-like

array of <=2 dimensions

`tol` : {None, float}
threshold below which SVD values are considered zero. If \( tol \) is None, and \( S \) is an array with singular values for \( M \), and \( \text{eps} \) is the epsilon value for datatype of \( S \), then \( tol \) set to 
\( S.\text{max()} \times \text{eps} \times \text{max}(M.\text{shape}) \).

**Notes**

We check for numerical rank deficiency by using \( tol=\text{max}(M.\text{shape}) \times \text{eps} \times S[0] \) (where \( S[0] \) is the maximum singular value and thus the 2-norm of the matrix). This is one tolerance threshold for rank deficiency, and the default algorithm used by MATLAB [2]. When floating point roundoff is the main concern, then “numerical rank deficiency” is a reasonable choice. In some cases you may prefer other definitions. The most useful measure of the tolerance depends on the operations you intend to use on your matrix. For example, if your data come from uncertain measurements with uncertainties greater than floating point epsilon, choosing a tolerance near that uncertainty may be preferable. The tolerance may be absolute if the uncertainties are absolute rather than relative.

**References**


http://www.mathworks.com/help/techdoc/ref/rank.html

[R10], [R11]

**Examples**

```python
>>> matrix_rank(np.eye(4))  # Full rank matrix
4
>>> I=np.eye(4); I[-1,-1] = 0.  # rank deficient matrix
>>> matrix_rank(I)
3
>>> matrix_rank(np.zeros((4,4)))  # All zeros - zero rank
0
>>> matrix_rank(np.ones((4,)))  # 1 dimension - rank 1 unless all 0
1
>>> matrix_rank(np.zeros((4,)))
0
>>> matrix_rank([1])  # accepts array-like
1
```

**nipy.algorithms.utils.matrices.pos_recipr(X)**

Return element-wise reciprocal of array, setting \( X \geq 0 \) to 0

Return the reciprocal of an array, setting all entries less than or equal to 0 to 0. Therefore, it presumes that \( X \) should be positive in general.

**Parameters**

\( X \): array-like

**Returns**

\( rX \): array

array of same shape as \( X \), dtype np.float, with values set to \( 1/X \) where \( X > 0 \), 0 otherwise

**nipy.algorithms.utils.matrices.recipr0(X)**

Return element-wise reciprocal of array, ‘\( X\)'\( \geq 0 \) -> 0

Return the reciprocal of an array, setting all entries equal to 0 as 0. It does not assume that \( X \) should be positive in general.
Parameters  $X$ : array-like

Returns  $rX$ : array
68.1 Module: algorithms.utils.pca

This module provides a class for principal components analysis (PCA).

PCA is an orthonormal, linear transform (i.e., a rotation) that maps the data to a new coordinate system such that the maximal variability of the data lies on the first coordinate (or the first principal component), the second greatest variability is projected onto the second coordinate, and so on. The resulting data has unit covariance (i.e., it is decorrelated). This technique can be used to reduce the dimensionality of the data.

More specifically, the data is projected onto the eigenvectors of the covariance matrix.

68.2 Functions

nipy.algorithms.utils.pca.pca(data, axis=0, mask=None, ncomp=None, standardize=True, design_keep=None, design_resid='mean', tol_ratio=0.01)

Compute the SVD PCA of an array-like thing over axis.

Parameters

data : ndarray-like (np.float)
The array on which to perform PCA over axis axis (below)

axis : int, optional
The axis over which to perform PCA (axis identifying observations). Default is 0 (first)

mask : ndarray-like (np.bool), optional
An optional mask, should have shape given by data axes, with axis removed, i.e.: s = data.shape; s.pop(axis); msk_shape=s

ncomp : {None, int}, optional
How many component basis projections to return. If ncomp is None (the default) then the number of components is given by the calculated rank of the data, after applying design_keep, design_resid and tol_ratio below. We always return all the basis vectors and percent variance for each component; ncomp refers only to the number of basis_projections returned.

standardize : bool, optional
If True, standardize so each time series (after application of design_keep and design_resid) has the same standard deviation, as calculated by the np.std function.

design_keep : None or ndarray, optional
Data is projected onto the column span of design_keep. None (default) equivalent to np.identity(data.shape[axis])

design_resid : str or None or ndarray, optional

After projecting onto the column span of design_keep, data is projected perpendicular to the column span of this matrix. If None, we do no such second projection. If a string ‘mean’, then the mean of the data is removed, equivalent to passing a column vector matrix of 1s.

tol_ratio : float, optional

If $XZ$ is the vector of singular values of the projection matrix from design_keep and design_resid, and $S$ are the singular values of $XZ$, then tol_ratio is the value used to calculate the effective rank of the projection of the design, as in $\text{rank} = (S / S.\text{max}) > \text{tol_ratio}).\text{sum}()$

Returns results : dict

$G$ is the number of non-trivial components found after applying tol_ratio to the projections of design_keep and design_resid.

results has keys:

• basis_vectors: series over axis, shape (data.shape[axis], G) - the eigenvectors of the PCA

• pcnt_var: percent variance explained by component, shape (G,)

• basis_projections: PCA components, with components varying over axis axis; thus shape given by: s = list(data.shape); s[axis] = ncomp

• axis: axis over which PCA has been performed.

Notes

See pca_image.m from fmristat for Keith Worsley’s code on which some of this is based.

See: http://en.wikipedia.org/wiki/Principal_component_analysis for some inspiration for naming - particularly ‘basis_vectors’ and ‘basis_projections’

Examples

```python
>>> arr = np.random.normal(size=(17, 10, 12, 14))
>>> msk = np.all(arr > -2, axis=0)
>>> res = pca(arr, mask=msk, ncomp=9)
```

Basis vectors are columns. There is one column for each component. The number of components is the calculated rank of the data matrix after applying the various projections listed in the parameters. In this case we are only removing the mean, so the number of components is one less than the axis over which we do the PCA (here axis=0 by default).

```python
>>> res[‘basis_vectors’].shape
(17, 16)
```

Basis projections are arrays with components in the dimension over which we have done the PCA (axis=0 by default). Because we set ncomp above, we only retain ncomp components.
Compute the PCA of an image over a specified axis

**Parameters**

- **img** : Image
  - The image on which to perform PCA over the given *axis*

- **axis** : str or int
  - Axis over which to perform PCA. Default is ‘t’. If *axis* is an integer, gives the index of the input (domain) axis of *img*. If *axis* is a str, can be an input (domain) name, or an output (range) name, that maps to an input (domain) name.

- **mask** : Image, optional
  - An optional mask, should have shape == image.shape[:3] and the same coordinate map as *img* but with *axis* dropped

- **ncomp** : {None, int}, optional
  - How many component basis projections to return. If ncomp is None (the default) then the number of components is given by the calculated rank of the data, after applying *design_keep*, *design_resid* and *tol_ratio* below. We always return all the basis vectors and percent variance for each component; *ncomp* refers only to the number of basis_projections returned.

- **standardize** : bool, optional
  - If True, standardize so each time series (after application of *design_keep* and *design_resid*) has the same standard deviation, as calculated by the `np.std` function.

- **design_keep** : None or ndarray, optional
  - Data is projected onto the column span of *design_keep*. None (default) equivalent to `np.identity(data.shape[axis])`

- **design_resid** : str or None or ndarray, optional
  - After projecting onto the column span of *design_keep*, data is projected perpendicular to the column span of this matrix. If None, we do no such second projection. If a string ‘mean’, then the mean of the data is removed, equivalent to passing a column vector matrix of 1s.

- **tol_ratio** : float, optional
  - If $X Z$ is the vector of singular values of the projection matrix from *design_keep* and *design_resid*, and $S$ are the singular values of $X Z$, then *tol_ratio* is the value used to calculate the effective rank of the projection of the design, as in $\\text{rank} = ((S / S.\text{max}) > \text{tol_ratio}).\text{sum()}$.

**Returns**

- **results** : dict
  - $L$ is the number of non-trivial components found after applying *tol_ratio* to the projections of *design_keep* and *design_resid*.
  - *results* has keys: *basis_vectors*: series over *axis*, shape (data.shape[axis], L) - the eigenvectors of the PCA
• `pcnt_var`: percent variance explained by component, shape \((L,)\)

• `basis_projections`: PCA components, with components varying over axis 
  \(axis\); thus shape given by: \(s = \text{list(data.shape)}; s[axis] = \text{ncomp}\)

• `axis`: axis over which PCA has been performed.

Examples

```python
>>> from nipy.testing import funcfile
>>> from nipy import load_image
>>> func_img = load_image(funcfile)

Time is the fourth axis

>>> func_img.coordmap.function_range
CoordinateSystem(coord_names=('aligned-x=L->R', 'aligned-y=P->A', 'aligned-z=I->S', 't'), name='aligned', coord_dtype=float64)

>>> func_img.shape
(17, 21, 3, 20)

Calculate the PCA over time, by default

>>> res = pca_image(func_img)

>>> res['basis_projections'].coordmap.function_range
CoordinateSystem(coord_names=('aligned-x=L->R', 'aligned-y=P->A', 'aligned-z=I->S', 'PCA components'), name='aligned', coord_dtype=float64)

The number of components is one less than the number of time points

>>> res['basis_projections'].shape
(17, 21, 3, 19)
```
69.1 Module: `core.image.image`

Inheritance diagram for `nipy.core.image.image`:

```
image.image.Image
image.image.SliceMaker
```

Define the Image class and functions to work with Image instances

- `fromarray` : create an Image instance from an ndarray (deprecated in favor of using the Image constructor)
- `subsample` : slice an Image instance (deprecated in favor of image slicing)
- `rollaxis` : roll an image axis backwards
- `synchronized_order` : match coordinate systems between images
- `iter_axis` : make iterator to iterate over an image axis
- `is_image` : test for an object obeying the Image API

69.2 Classes

69.2.1 Image

```python
class nipy.core.image.image.Image(data, coordmap, metadata=None)
Bases: object
```

The `Image` class provides the core object type used in nipy.

An `Image` represents a volumetric brain image and provides means for manipulating the image data. Most functions in the image module operate on `Image` objects.
Notes

Images can be created through the module functions. See nipy.io for image IO such as load and save.

Examples

Load an image from disk

```python
>>> from nipy.testing import anatfile
>>> from nipy.io.api import load_image

>>> img = load_image(anatfile)
```

Make an image from an array. We need to make a meaningful coordinate map for the image.

```python
>>> arr = np.zeros((21,64,64), dtype=np.int16)
>>> cmap = AffineTransform('kji', 'zxy', np.eye(4))
>>> img = Image(arr, cmap)
```

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>affine()</td>
<td></td>
</tr>
<tr>
<td>axes()</td>
<td></td>
</tr>
<tr>
<td>coordmap</td>
<td>Class for affine transformation from domain to a range</td>
</tr>
<tr>
<td>from_image(klass, img, ..., data, coordmap, ...])</td>
<td>Classmethod makes new instance of this klass from instance img</td>
</tr>
<tr>
<td>get_data()</td>
<td>Return data as a numpy array.</td>
</tr>
<tr>
<td>ndim()</td>
<td></td>
</tr>
<tr>
<td>reference()</td>
<td></td>
</tr>
<tr>
<td>renamed_axes(**names_dict)</td>
<td>Return a new image with input (domain) axes renamed</td>
</tr>
<tr>
<td>renamed_reference(**names_dict)</td>
<td>Return new image with renamed output (range) coordinates</td>
</tr>
<tr>
<td>reordered_axes(order)</td>
<td>Return a new Image with reordered input coordinates.</td>
</tr>
<tr>
<td>reordered_reference(order)</td>
<td>Return new Image with reordered output coordinates.</td>
</tr>
<tr>
<td>shape()</td>
<td></td>
</tr>
</tbody>
</table>

__init__(data, coordmap, metadata=None)

Create an Image object from array and CoordinateMap object.

Images are often created through the load_image function in the nipy base namespace.

**Parameters**

- **data**: array-like object that as attribute shape and returns an array from np.asarray(data)
- **coordmap**: AffineTransform object
  - coordmap mapping the domain (input) voxel axes of the image to the range (reference, output) axes - usually mm in real world space
- **metadata**: dict, optional
  - Freeform metadata for image. Most common contents is header from nifti etc loaded images.

See Also:

- load_image load Image from a file
save_image  save Image to a file

static affine()
static axes()

coordmap = AffineTransform( function_domain=CoordinateSystem(coord_names=('i', 'j', 'k'), name='', coord_dtype=float64),
                            function_range=CoordinateSystem(coord_names=('a', 'b', 1), name='', coord_dtype=float64))

classmethod from_image( klass, img, data=None, coordmap=None, metadata=None)
    Classmethod makes new instance of this klass from instance img

Parameters
data : array-like
    object that as attribute shape and returns an array from np.asarray(data)

coordmap : AffineTransform object
    coordmap mapping the domain (input) voxel axes of the image to the range (reference, output) axes - usually mm in real world space

metadata : dict, optional
    Freeform metadata for image. Most common contents is header from nifti etc loaded images.

Returns
    img : klass instance
    New image with data from data, coordmap from coordmap maybe metadata from metadata

Notes
Subclasses of Image with different semantics for __init__ will need to override this classmethod.

Examples

>>> from nipy import load_image
>>> from nipy.core.api import Image
>>> from nipy.testing import anatfile

>>> aimg = load_image(anatfile)
>>> arr = np.arange(24).reshape((2,3,4))

>>> img = Image.from_image(aimg, data=arr)

get_data()
    Return data as a numpy array.

header
    The file header structure for this image, if available. This interface will soon go away - you should use 
    "img.metadata['header'] instead.

metadata = {}
static ndim()
static reference()
renamed_axes(**names_dict)
    Return a new image with input (domain) axes renamed

    Axes renamed according to the input dictionary.

Parameters
    **names_dict : dict
with keys being old names, and values being new names

Returns  **newimg** : Image

An Image with the same data, having its axes renamed.

Examples

```python
>>> data = np.random.standard_normal((11, 9, 4))
>>> im = Image(data, AffineTransform.from_params('ijk', 'xyz', np.identity(4), 'domain', 'range')
>>> im_renamed = im.renamed_axes(i='slice')
>>> print im_renamed.axes
CoordinateSystem(coord_names=('slice', 'j', 'k'), name='domain', coord_dtype=float64)
```

renamed_reference (**names_dict**)

Return new image with renamed output (range) coordinates

Coordinates renamed according to the dictionary

Parameters  **names_dict** : dict

with keys being old names, and values being new names

Returns  **newimg** : Image

An Image with the same data, having its output coordinates renamed.

Examples

```python
>>> data = np.random.standard_normal((11, 9, 4))
>>> im = Image(data, AffineTransform.from_params('ijk', 'xyz', np.identity(4), 'domain', 'range')
>>> im_renamed_reference = im.renamed_reference(x='newx', y='newy')
>>> print im_renamed_reference.reference
CoordinateSystem(coord_names=('newx', 'newy', 'z'), name='range', coord_dtype=float64)
```

reordered_axes (**order**=None)

Return a new Image with reordered input coordinates.

This transposes the data as well.

Parameters  **order** : None, sequence, optional

Sequence of int (giving indices) or str (giving names) - expressing new order of coordmap output coordinates. None (the default) results in reversed ordering.

Returns  **r_img** : object

Image of same class as self, with reordered output coordinates.

Examples

```python
>>> cmap = AffineTransform.from_start_step(
...   'ijk', 'xyz', 1, 2, 3, [4, 5, 6], 'domain', 'range')
>>> cmap
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k'), name='domain', coord_dtype=float64)
    function_range=CoordinateSystem(coord_names=('x', 'y', 'z'), name='range', coord_dtype=float64)
    affine=array([[ 4., 0., 0., 1.],
```

```python
```
>>> im = Image(np.empty((30,40,50)), cmap)
>>> im_reordered = im.reordered_axes([2,0,1])
>>> im_reordered.shape
(50, 30, 40)
>>> im_reordered.coordmap
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('k', 'i', 'j'), name='domain', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('x', 'y', 'z'), name='range', coord_dtype=float64),
    affine=array([[ 0., 4., 0., 1.],
                   [ 0., 0., 5., 2.],
                   [ 6., 0., 0., 3.],
                   [ 0., 0., 0., 1.]]))

reordered_reference(order=None)

Return new Image with reordered output coordinates

New Image coordmap has reordered output coordinates. This does not transpose the data.

Parameters

order : None, sequence, optional

sequence of int (giving indices) or str (giving names) - expressing new order of coordmap output coordinates. None (the default) results in reversed ordering.

Returns

r_img : object

Image of same class as self, with reordered output coordinates.

Examples

>>> cmap = AffineTransform.from_start_step(...
    'ijk', 'xyz', [1, 2, 3], [4, 5, 6], 'domain', 'range')
>>> im = Image(np.empty((30,40,50)), cmap)
>>> im_reordered = im.reordered_reference([2,0,1])
>>> im_reordered.shape
(30, 40, 50)
>>> im_reordered.coordmap
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k'), name='domain', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('z', 'x', 'y'), name='range', coord_dtype=float64),
    affine=array([[ 0., 0., 6., 3.],
                  [ 4., 0., 0., 1.],
                  [ 0., 5., 0., 2.],
                  [ 0., 0., 0., 1.]]))

static shape()

69.2.2 SliceMaker

class nipy.core.image.image.SliceMaker

Bases: object

This class just creates slice objects for image resampling
It only has a \_getitem\_ method that returns its argument.

XXX Wouldn’t need this if there was a way XXX to do this XXX subsample(img, [::2::3,10:1:-1]) XXX
Could be something like this Subsample(img)[::2::3,10:1:-1]

\_init\_(())
  x._\_init\_(...) initializes x; see help(type(x)) for signature

## 69.3 Functions

nipy.core.image.image.fromarray(data, innames, outnames)

Create an image from array data, and input/output coordinate names

The mapping between the input and output coordinate names is the identity matrix.

Please don’t use this routine, but instead prefer:

```python
from nipy.core.api import Image, AffineTransform
img = Image(data, AffineTransform(innames, outnames, np.eye(4)))
```

where 4 is len(innames) + 1.

- **Parameters**
  - data: numpy array
    - A numpy array of three dimensions.
  - innames: sequence
    - a list of input axis names
  - outnames: sequence
    - a list of output axis names

- **Returns**
  - image: An Image object

See Also:

- load function for loading images
- save function for saving images

### Examples

```python
>>> img = fromarray(np.zeros((2,3,4)), 'ijk', 'xyz')
>>> img.coordmap
AffineTransform(
  function_domain=CoordinateSystem(coord_names=('i', 'j', 'k'), name='', coord_dtype=float64),
  function_range=CoordinateSystem(coord_names=('x', 'y', 'z'), name='', coord_dtype=float64),
  affine=array([[ 1.,  0.,  0.,  0.],
                [ 0.,  1.,  0.,  0.],
                [ 0.,  0.,  1.,  0.],
                [ 0.,  0.,  0.,  1.]])
)
```

nipy.core.image.image.is_image(obj)

Returns true if this object obeys the Image API

This allows us to test for something that is duck-typing an image.

For now an array must have a ‘coordmap’ attribute, and a callable ‘get_data’ attribute.
**Parameters**  
\texttt{obj}: object
  
object for which to test API

**Returns**  
\texttt{is_img}: bool
  
True if object obeys image API

**Examples**

```python
>>> from nipy.testing import anatfile
>>> from nipy.io.api import load_image
>>> img = load_image(anatfile)
>>> is_image(img)
True
>>> class C(object): pass
>>> c = C()
>>> is_image(c)
False
```

nipy.core.image.image.\texttt{iter_axis}(\texttt{img}, \texttt{axis}, \texttt{asarray}=\texttt{False})

Return generator to slice an image \texttt{img} over \texttt{axis}

**Parameters**  
\texttt{img}: Image instance

\texttt{axis}: int or str
  
axis identifier, either name or axis number

\texttt{asarray}: \{False, True\}, optional

**Returns**  
\texttt{g}: generator
  
such that list(g) returns a list of slices over \texttt{axis}. If \texttt{asarray} is \texttt{False} the slices are images. 
If \texttt{asarray} is \texttt{True}, slices are the data from the images.

**Examples**

```python
>>> data = np.arange(24).reshape((4,3,2))
>>> img = Image(data, AffineTransform('ijk', 'xyz', np.eye(4)))
>>> slices = list(iter_axis(img, 'j'))
>>> len(slices)
3
>>> slices[0].shape
(4, 2)
>>> slices[1].sum() == data[:,:,1].sum()
True
```

nipy.core.image.image.\texttt{rollaxis}(\*\texttt{args}, **\texttt{kwds})

\texttt{rollaxis} is deprecated! Please use \texttt{rolling} instead

Roll \texttt{axis} backwards, until it lies in the first position.

It also reorders the reference coordinates by the same ordering. This is done to preserve a diagonal affine matrix if image.affine is diagonal. It also makes it possible to unambiguously specify an axis to roll along in terms of either a reference name (i.e. ‘z’) or an axis name (i.e. ‘slice’).

This function is deprecated; please use \texttt{rolling} instead.
Parameters  

`img` : Image
Image whose axes and reference coordinates are to be reordered by rolling.

`axis` : str or int
Axis to be rolled, can be specified by name or as an integer.

`inverse` : bool, optional
If `inverse` is True, then `axis` must be an integer and the first axis is returned to the position axis. This keyword is deprecated and we’ll remove it in a future version of nipy.

Returns  

`newimg` : Image
Image with reordered axes and reference coordinates.

Examples

```python
>>> data = np.zeros((30,40,50,5))
>>> affine_transform = AffineTransform.from_params('ijkl', 'xyzt', np.diag([1,2,3,4,1]))
>>> im = Image(data, affine_transform)
>>> im.coordmap
AffineTransform(
  function_domain=CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='', coord_dtype=float64),
  function_range=CoordinateSystem(coord_names=('x', 'y', 'z', 't'), name='', coord_dtype=float64),
  affine=array([[ 1., 0., 0., 0., 0.,],
                 [ 0., 2., 0., 0., 0.,],
                 [ 0., 0., 3., 0., 0.,],
                 [ 0., 0., 0., 4., 0.,],
                 [ 0., 0., 0., 0., 1.]]))
>>> im_t_first = rollaxis(im, 't')
```

```python
>>> np.diag(im_t_first.affine)
array([ 4., 1., 2., 3., 1.])
```

```python
>>> im_t_first.shape
(5, 30, 40, 50)
```

```python
>>> im_t_first.coordmap
AffineTransform(
  function_domain=CoordinateSystem(coord_names=('l', 'i', 'j', 'k'), name='', coord_dtype=float64),
  function_range=CoordinateSystem(coord_names=('t', 'x', 'y', 'z'), name='', coord_dtype=float64),
  affine=array([[ 4., 0., 0., 0., 0.,],
                [ 0., 1., 0., 0., 0.,],
                [ 0., 0., 2., 0., 0.,],
                [ 0., 0., 0., 3., 0.],
                [ 0., 0., 0., 0., 1.]]))
```

nipy.core.image.rollimg (img, axis, start=0, fix0=True)
Roll `axis` backwards in the inputs, until it lies before `start`

Parameters  

`img` : Image
Image whose axes and reference coordinates are to be reordered by rollimg.

`axis` : str or int
Axis to be rolled, can be specified by name or as an integer. If an integer, `axis` is an input axis. If a name, can be name of input or output axis. If an output axis, we search for the closest matching input axis, and raise an AxisError if this fails.

`start` : str or int, optional
position before which to roll axis `axis`. Default to 0. Can again be an integer (input axis) or name of input or output axis.

**fix0**: bool, optional

Whether to allow for zero scaling when searching for an input axis matching an output axis. Useful for images where time scaling is 0.

**Returns**  newimg : Image

Image with reordered input axes and corresponding data.

**Examples**

```python
>>> data = np.zeros((30,40,50,5))
>>> affine_transform = AffineTransform('ijkl', 'xyzt', np.diag([1,2,3,4,1]))
>>> im = Image(data, affine_transform)
>>> im.coordmap
AffineTransform(  
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='', coord_dtype=float64),  
    function_range=CoordinateSystem(coord_names=('x', 'y', 'z', 't'), name='', coord_dtype=float64),  
    affine=array([[ 1., 0., 0., 0., 0.],  
                   [ 0., 2., 0., 0., 0.],  
                   [ 0., 0., 3., 0., 0.],  
                   [ 0., 0., 0., 4., 0.],  
                   [ 0., 0., 0., 0., 1.]]))

>>> im_t_first = rollimg(im, 't')
>>> im_t_first.shape
(5, 30, 40, 50)
>>> im_t_first.coordmap
AffineTransform(  
    function_domain=CoordinateSystem(coord_names=('l', 'i', 'j', 'k'), name='', coord_dtype=float64),  
    function_range=CoordinateSystem(coord_names=('x', 'y', 'z', 't'), name='', coord_dtype=float64),  
    affine=array([[ 0., 1., 0., 0., 0.],  
                   [ 0., 0., 2., 0., 0.],  
                   [ 0., 0., 0., 3., 0.],  
                   [ 4., 0., 0., 0., 0.],  
                   [ 0., 0., 0., 0., 1.]]))
```

```
nipy.core.image.image.subsample(img, slice_object)

Subsample an image

Please don’t use this function, but use direct image slicing instead. That is, replace:

```python
    frame3 = subsample(im, slice_maker[:,:,:,:])
```

with:

```python
    frame3 = im[:,:,:,:]
```

**Parameters**  img : Image

    slice_object: int, slice or sequence of slice :

    An object representing a numpy ‘slice’.

**Returns**  img_subsampled: Image :
An Image with data img.get_data()[slice_object] and an appropriately corrected CoordinateMap.

Examples

```python
>>> from nipy.io.api import load_image
>>> from nipy.testing import funcfile
>>> from nipy.core.api import subsample, slice_maker

>>> im = load_image(funcfile)

>>> frame3 = subsample(im, slice_maker[:,:,:,3])

>>> np.allclose(frame3.get_data(), im.get_data()[:,:,:,3])
True
```

**nipy.core.image.image.synchronized_order**(img, target_img, axes=True, reference=True)

Reorder reference and axes of `img` to match `target_img`.

**Parameters**

- `img` : Image
- `target_img` : Image
- `axes` : bool, optional
  - If True, synchronize the order of the axes.
- `reference` : bool, optional
  - If True, synchronize the order of the reference coordinates.

**Returns**

- `newimg` : Image


Examples

```python
>>> data = np.random.standard_normal((3,4,7,5))

>>> im = Image(data, AffineTransform.from_params('ijkl', 'xyzt', np.diag([1,2,3,4,1])))

>>> im_scrambled = im.reordered_axes('iljk').reordered_reference('txyz')

>>> im == im_scrambled
False

>>> im_unscrambled = synchronized_order(im_scrambled, im)

>>> im == im_unscrambled
True
```

The images don’t have to be the same shape

```python
>>> data2 = np.random.standard_normal((3,11,9,4))

>>> im2 = Image(data2, AffineTransform.from_params('ijkl', 'xyzt', np.diag([1,2,3,4,1])))

>>> im_unscrambled2 = synchronized_order(im_scrambled2, im)

>>> im_unscrambled2.coordmap == im.coordmap
True
```

or have the same coordmap

```python
>>> data3 = np.random.standard_normal((3,11,9,4))

>>> im3 = Image(data3, AffineTransform.from_params('ijkl', 'xyzt', np.diag([1,9,3,-2,1])))

>>> im_scrambled3 = im3.reordered_axes('iljk').reordered_reference('txyz')

>>> im_unscrambled3 = synchronized_order(im_scrambled3, im)
```
>>> im_unscrambled3.axes == im.axes
True
>>> im_unscrambled3.reference == im.reference
True
>>> im_unscrambled4 = synchronized_order(im_scrambled3, im, axes=False)
>>> im_unscrambled4.axes == im.axes
False
>>> im_unscrambled4.axes == im_scrambled3.axes
True
>>> im_unscrambled4.reference == im.reference
True
70.1 Module: core.image.image_list

Inheritance diagram for nipy.core.image.image_list:

```
image.image_list.ImageList
```

70.2 ImageList

class nipy.core.image.image_list.ImageList (images=None)
  Bases: object
  
  Class to contain ND image as list of (N-1)D images

  Methods

  from_image(klass, image[, axis, dropout])  Create an image list from an image by slicing over axis
  get_list_data([axis])  Return data in ndarray with list dimension at position axis
  next()

__init__ (images=None)
  An implementation of a list of images.

  Parameters  images : iterable

  an iterable object whose items are meant to be images; this is checked by asserting that
each has a coordmap attribute and a get_data method. Note that Image objects are
not iterable by default; use the from_image classmethod or iter_axis function to
convert images to image lists - see examples below for the latter.
Examples

```python
>>> from nipy.testing import funcfile
>>> from nipy.core.api import Image, ImageList, iter_axis
>>> from nipy.io.api import load_image

>>> funcim = load_image(funcfile)
>>> iterable_img = iter_axis(funcim, 't')
>>> ilist = ImageList(iterable_img)
>>> sublist = ilist[2:5]

Slicing an ImageList returns a new ImageList

>>> isinstance(sublist, ImageList)
True

Indexing an ImageList returns a new Image

>>> newimg = ilist[2]
>>> isinstance(newimg, Image)
True
>>> isinstance(newimg, ImageList)
False
>>> np.asarray(sublist).shape
(3, 17, 21, 3)
>>> newimg.get_data().shape
(17, 21, 3)
```

classmethod `from_image` *(klass, image, axis=None, dropout=True)*

Create an image list from an `image` by slicing over `axis`

**Parameters**

- `image` : object
  - object with `coordmap` attribute
- `axis` : str or int
  - axis of `image` that should become the axis indexed by the image list.
- `dropout` : bool, optional
  - When taking slices from an image, we will leave an output dimension to the coordmap that has no corresponding input dimension. If `dropout` is True, drop this output dimension.

**Returns**

- `ilist` : ImageList instance

`get_list_data` *(axis=None)*

Return data in ndarray with list dimension at position `axis`

**Parameters**

- `axis` : int
  - `axis` specifies which axis of the output will take the role of the list dimension. For example, 0 will put the list dimension in the first axis of the result.

**Returns**

- `data` : ndarray
  - data in image list as array, with data across elements of the list concatenated at dimension `axis` of the array.
Examples

```python
>>> from nipy.testing import funcfile
>>> from nipy.io.api import load_image
>>> funcim = load_image(funcfile)
>>> ilist = ImageList.from_image(funcim, axis='t')
>>> ilist.get_list_data(axis=0).shape
(20, 17, 21, 3)
```

`next()`
Utilities for working with Images and common neuroimaging spaces

Images are very general things, and don’t know anything about the kinds of spaces they refer to, via their coordinate map.

There are a set of common neuroimaging spaces. When we create neuroimaging Images, we want to place them in neuroimaging spaces, and return information about common neuroimaging spaces.

We do this by putting information about neuroimaging spaces in functions and variables in the nipy.core.reference.spaces module, and in this module.

This keeps the specific neuroimaging spaces out of our Image object.

```python
>>> from nipy.core.api import Image, vox2mni, rollimg, xyz_affine, as_xyz_image
```

Make a standard 4D xyzt image in MNI space.

First the data and affine:

```python
>>> data = np.arange(24).reshape((1,2,3,4))
>>> affine = np.diag([2,3,4,1]).astype(float)
```

We can add the TR (==2.0) to make the full 5x5 affine we need

```python
>>> img = Image(data, vox2mni(affine, 2.0))
```

In this case the neuroimaging `xyz_affine` is just the 4x4 from the 5x5 in the image

```python
>>> xyz_affine(img)
array([[ 2., 0., 0., 0.],
       [ 0., 3., 0., 0.],
       [ 0., 0., 4., 0.],
       [ 0., 0., 0., 1.]])
```

However, if we roll time first in the image array, we can’t any longer get an `xyz_affine` that makes sense in relationship to the voxel data:
>>> img_t0 = rollimg(img, 't')
>>> xyz_affine(img_t0)
Traceback (most recent call last):
...
AxesError: First 3 input axes must correspond to X, Y, Z

But we can fix this:

>>> img_t0_affable = as_xyz_image(img_t0)
>>> xyz_affine(img_t0_affable)
array([[2., 0., 0., 0.],
       [0., 3., 0., 0.],
       [0., 0., 4., 0.],
       [0., 0., 0., 1.]])

It also works with nibabel images, which can only have xyz_affines:

>>> import nibabel as nib
>>> nimg = nib.Nifti1Image(data, affine)
>>> xyz_affine(nimg)
array([[2., 0., 0., 0.],
       [0., 3., 0., 0.],
       [0., 0., 4., 0.],
       [0., 0., 0., 1.]])

## 71.2 Functions

nipy.core.image.image_spaces.as_xyz_image(img, name2xyz=None)

Return version of `img` that has a valid xyz affine, or raise error

**Parameters**

- `img`: Image instance or nibabel image
  - It has a `coordmap` attribute (Image) or a `get_affine` method (nibabel image object)
- `name2xyz`: None or mapping
  - Object such that `name2xyz[ax_name]` returns ‘x’, ‘y’ or ‘z’ or raises a KeyError for a str `ax_name`. None means use module default. Not used for nibabel `img` input.

**Returns**

- `reo_img`: Image instance or nibabel image
  - Returns image of same type as `img` input. If necessary, `reo_img` has its data and coordmap changed to allow it to return an xyz affine. If `img` is already xyz affable we return the input unchanged (`img` is `reo_img`).

**Raises**

- `SpaceTypeError`: if `img` does not have an affine coordinate map
- `AxesError`: if not all of x, y, z recognized in `img coordmap` range
- `AffineError`: if axes dropped from the affine contribute to x, y, z

nipy.core.image.image_spaces.is_xyz_affable(img, name2xyz=None)

Return True if the image `img` has an xyz affine

**Parameters**

- `img`: Image or nibabel SpatialImage
  - If Image test `img.coordmap`. If a nibabel image, return True

528 Chapter 71. core.image.image_spaces
**name2xyz**: None or mapping

Object such that `name2xyz[ax_name]` returns ‘x’, or ‘y’ or ‘z’ or raises a KeyError for a str `ax_name`. None means use module default. Not used for nibabel `img` input.

**Returns**: `tf`: bool

True if `img` has an xyz affine, False otherwise

**Examples**

```python
>>> from nipy.core.api import vox2mni, Image, rollimg

```

```python
>>> arr = np.arange(24).reshape((2,3,4,1))
>>> img = Image(arr, vox2mni(np.diag([2,3,4,5,1])))
>>> img.coordmap
 AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='voxels', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S', 't'), name='mni', coord_dtype=float64),
    affine=array([[ 2., 0., 0., 0., 0.],
                  [ 0., 3., 0., 0., 0.],
                  [ 0., 0., 4., 0., 0.],
                  [ 0., 0., 0., 5., 0.],
                  [ 0., 0., 0., 0., 1.]]))
>>> is_xyz_affable(img)
True
```

```python
>>> time0_img = rollimg(img, 't')
>>> time0_img.coordmap
 AffineTransform(
    function_domain=CoordinateSystem(coord_names=('l', 'i', 'j', 'k'), name='voxels', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S', 't'), name='mni', coord_dtype=float64),
    affine=array([[ 0., 2., 0., 0., 0.],
                  [ 0., 0., 3., 0., 0.],
                  [ 0., 0., 0., 4., 0.],
                  [ 5., 0., 0., 0., 0.],
                  [ 0., 0., 0., 0., 1.]]))
>>> is_xyz_affable(time0_img)
False
```

Nibabel images always have xyz affines

```python
>>> import nibabel as nib
>>> nimg = nib.Nifti1Image(arr, np.diag([2,3,4,1]))
>>> is_xyz_affable(nimg)
True
```

**nipy.core.image.image_spaces.make_xyz_image**(data, xyz_affine, world, metadata=None)

Create 3D+ image embedded in space named in `world`

**Parameters**

- **data**: object
  
  Object returning array from `np.asarray(obj)`, and having `shape` attribute. Should have at least 3 dimensions (`len(shape) >= 3`), and these three first 3 dimensions should be spatial

- **xyz_affine**: (4, 4) array-like or tuple
  
  if (4, 4) array-like (the usual case), then an affine relating spatial dimensions in data (dimensions 0:3) to mm in XYZ space given in `world`. If a tuple, then contains two
values: the (4, 4) array-like, and a sequence of scalings for the dimensions greater than 3. See examples.

world : str or XYZSpace or CoordSysMaker or CoordinateSystem
World 3D space to which affine refers. See spaces.get_world_cs()

metadata : None or mapping, optional
metadata for created image. Defaults to None, giving empty metadata.

Returns  img : Image
image containing data, with coordmap constructed from affine and world, and with default voxel input coordinates. If the data has more than 3 dimensions, and you didn’t specify the added zooms with a tuple xyz_affine parameter, the coordmap affine gets filled out with extra ones on the diagonal to give an (N+1, N+1) affine, with N = len(data.shape)

Examples

```python
>>> data = np.arange(24).reshape((2, 3, 4))
>>> aff = np.diag([4, 5, 6, 1])
>>> img = make_xyz_image(data, aff, 'mni')
>>> img
Image(
  data= array([[ 0, 1, 2, 3],
               [ 4, 5, 6, 7],
               [ 8, 9, 10, 11],
               [12, 13, 14, 15],
               [16, 17, 18, 19],
               [20, 21, 22, 23]]),
  coordmap= AffineTransform(  
  function_domain= CoordinateSystem(coord_names=('i', 'j', 'k'), name='voxels', coord_dtype=float64),
  function_range= CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S'), name='mni', coord_dtype=float64),
  affine= array([[ 4., 0., 0., 0.],
               [ 0., 5., 0., 0.],
               [ 0., 0., 6., 0.],
               [ 0., 0., 0., 1.]]),
  )
)
```

Now make data 4D; we just add 1. to the diagonal for the new dimension

```python
>>> data4 = data[..., None]
>>> img = make_xyz_image(data4, aff, 'mni')
>>> img.coordmap
AffineTransform(  
  function_domain= CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='voxels', coord_dtype=float64),
  function_range= CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S', 't'), name='mni', coord_dtype=float64),
  affine= array([[ 4., 0., 0., 0., 0.],
               [ 0., 5., 0., 0., 0.],
               [ 0., 0., 6., 0., 0.],
               [ 0., 0., 0., 1., 0.],
               [ 0., 0., 0., 0., 1.]]),
  )
```

We can pass in a scalar or tuple to specify scaling for the extra dimension
```python
>>> img = make_xyz_image(data4, (aff, 2.0), 'mni')
>>> img.coordmap.affine
array([[ 4., 0., 0., 0., 0.],
       [ 0., 5., 0., 0., 0.],
       [ 0., 0., 6., 0., 0.],
       [ 0., 0., 0., 2., 0.],
       [ 0., 0., 0., 0., 1.]]
>>> data5 = data4[... , None]
>>> img = make_xyz_image(data5, (aff, (2.0, 3.0)), 'mni')
>>> img.coordmap.affine
array([[ 4., 0., 0., 0., 0., 0.],
       [ 0., 5., 0., 0., 0., 0.],
       [ 0., 0., 6., 0., 0., 0.],
       [ 0., 0., 0., 2., 0., 0.],
       [ 0., 0., 0., 0., 3., 0.],
       [ 0., 0., 0., 0., 0., 1.]]
```

```python
nipy.core.image.image_spaces.xyz_affine(img, name2xyz=None)
```

Return xyz affine from image `img` if possible, or raise error

**Parameters**

- **img**: Image instance or nibabel image
  - It has a coordmap or method `get_affine`
- **name2xyz**: None or mapping
  - Object such that `name2xyz[ax_name]` returns ‘x’, or ‘y’ or ‘z’ or raises a KeyError for a str `ax_name`. None means use module default. Not used for nibabel `img` input.

**Returns**

- **xyz_aff**: (4,4) array
  - Voxel to X, Y, Z affine mapping

**Raises**

- **SpaceTypeError**: if `img` does not have an affine coordinate map
- **AxesError**: if not all of x, y, z recognized in `img coordmap range`
- **AffineError**: if axes dropped from the affine contribute to x, y, z coordinates:

**Examples**

```python
>>> from nipy.core.api import vox2mni, Image
>>> arr = np.arange(24).reshape((2,3,4,1)).astype(float)
>>> img = Image(arr, vox2mni(np.diag([2,3,4,5,1])))
>>> img.coordmap
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='voxels', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S', 't'),
    affine=array([[ 2., 0., 0., 0., 0.],
       [ 0., 3., 0., 0., 0.],
       [ 0., 0., 4., 0., 0.],
       [ 0., 0., 0., 5., 0.],
       [ 0., 0., 0., 0., 1.]]
    )
)
```
```
Nibabel images always have xyz affines

```python
>>> import nibabel as nib
>>> nimg = nib.Nifti1Image(arr, np.diag([2,3,4,1]))
>>> xyz_affine(nimg)
array([[ 2., 0., 0., 0.],
       [ 0., 3., 0., 0.],
       [ 0., 0., 4., 0.],
       [ 0., 0., 0., 1.]])
```
72.1 Module: core.reference.array_coords

Inheritance diagram for nipy.core.reference.array_coords:

- reference.array_coords.Grid
- reference.array_coords.ArrayCoordMap

Some CoordinateMaps have a domain that are ‘array’ coordinates, hence the function of the CoordinateMap can be evaluated at these ’array’ points.

This module tries to make these operations easier by defining a class ArrayCoordMap that is essentially a CoordinateMap and a shape.

This class has two properties: values, transposed_values the CoordinateMap at np.indices(shape).

The class Grid is meant to take a CoordinateMap and an np.mgrid-like notation to create an ArrayCoordMap.

72.2 Classes

72.2.1 ArrayCoordMap

class nipy.core.reference.array_coords.ArrayCoordMap(coordmap, shape):
    Bases: object
    
    Class combining coordinate map and array shape
    
    When the function_domain of a CoordinateMap can be thought of as ‘array’ coordinates, i.e. an ‘input_shape’ makes sense. We can than evaluate the CoordinateMap at np.indices(input_shape)
Methods

```
from_shape(coordmap, shape)  # Create an evaluator assuming that coordmap.function_domain

__init__(coordmap, shape)

Parameters:
coordmap : CoordinateMap
A CoordinateMap with function_domain that are ‘array’ coordinates.
shape : sequence of int
The size of the (implied) underlying array.
```

Examples

```python
def main():
    aff = np.diag([0.6,1.1,2.3,1])
    aff[:3,3] = (0.1, 0.2, 0.3)
    cmap = AffineTransform.from_params('ijk', 'xyz', aff)
    cmap = ArrayCoordMap(cmap, (1, 2, 3))
    Real world values at each array coordinate, one row per array coordinate (6 in this case), one column for each output dimension (3 in this case)
    >>> cmap.values
    array([[ 0.1, 0.2, 0.3],
           [ 0.1, 0.2, 2.6],
           [ 0.1, 0.2, 4.9],
           [ 0.1, 1.3, 0.3],
           [ 0.1, 1.3, 2.6],
           [ 0.1, 1.3, 4.9]])

    Same values, but arranged in np.indices / np.mgrid format, first axis is for number of output coordinates (3 in our case), the rest are for the input shape (1, 2, 3)
    >>> cmap.transposed_values
    array([[ 0.1, 0.1, 0.1],
           [ 0.1, 0.1, 0.1],
           [ 0.2, 0.2, 0.2],
           [ 1.3, 1.3, 1.3],
           [ 0.3, 2.6, 4.9],
           [ 0.3, 2.6, 4.9]])

    static from_shape(coordmap, shape)
    Create an evaluator assuming that coordmap.function_domain are ‘array’ coordinates.
    transposed_values
    Get values of ArrayCoordMap in an array of shape (self.coordmap.ndims[1],) + self.shape)
```
values
Get values of ArrayCoordMap in a 2-dimensional array of shape (product(self.shape), self.coordmap.ndims[1])

72.2.2 Grid

class nipy.core.reference.array_coords.Grid(coords)
    Bases: object

    Simple class to construct AffineTransform instances with slice notation like np.ogrid/np.mgrid.

>>> c = CoordinateSystem('xy', 'input')
>>> g = Grid(c)
>>> points = g[-1:1:21j,-2:4:31j]
>>> points.coordmap.affine
array([[ 0.1, 0., -1.],
       [ 0., 0.2, -2.],
       [ 0., 0., 1.]]

>>> print points.coordmap.function_domain
CoordinateSystem(coord_names=('i0', 'i1'), name='product', coord_dtype=float64)
>>> print points.coordmap.function_range
CoordinateSystem(coord_names=('x', 'y'), name='input', coord_dtype=float64)

>>> points.shape
(21, 31)
>>> print points.transposed_values.shape
(2, 21, 31)
>>> print points.values.shape
(651, 2)

__init__(coords)
Initialize Grid object

    Parameters coords: "CoordinateMap" or "CoordinateSystem":

    A coordinate map to be 'sliced' into. If coords is a CoordinateSystem, then an Affine-
    Transform instance is created with coords with identity transformation.
73.1 Module: core.reference.coordinate_map

This module describes two types of mappings:

- **CoordinateMap**: a general function from a domain to a range, with a possible inverse function.
- **AffineTransform**: an affine function from a domain to a range, not necessarily of the same dimension, hence not always invertible.

Each of these objects is meant to encapsulate a tuple of (domain, range, function). Each of the mapping objects contain all the details about their domain CoordinateSystem, their range CoordinateSystem and the mapping between them.

73.1.1 Common API

They are separate classes, neither one inheriting from the other. They do, however, share some parts of an API, each having methods:
• renamed_domain : rename on the coordinates of the domain (returns a new mapping)
• renamed_range : rename the coordinates of the range (returns a new mapping)
• reordered_domain : reorder the coordinates of the domain (returns a new mapping)
• reordered_range : reorder the coordinates of the range (returns a new mapping)
• inverse : when appropriate, return the inverse mapping

These methods are implemented by module level functions of the same name.

They also share some attributes:
• ndims : the dimensions of the domain and range, respectively
• function_domain : CoordinateSystem describing the domain
• function_range : CoordinateSystem describing the range

73.1.2 Operations on mappings (module level functions)

• compose [Take a sequence of mappings (either CoordinateMaps or AffineTransforms) and return their composition. If they are all AffineTransforms, an AffineTransform is returned. This checks to ensure that domains and ranges of the various mappings agree.

• product [Take a sequence of mappings (either CoordinateMaps or AffineTransforms) and return a new mapping that has domain and range given by the concatenation of their domains and ranges, and the mapping simply concatenates the output of each of the individual mappings. If they are all AffineTransforms, an AffineTransform is returned. If they are all AffineTransforms that are in fact linear (i.e. origin=0) then can is represented as a block matrix with the size of the blocks determined by

• concat [Take a mapping and prepend a coordinate to its domain and] range. For mapping m, this is the same as product(AffineTransform.identity('concat'), m)

73.2 Classes

73.2.1 AffineTransform

class nipy.core.reference.coordinate_map.AffineTransform(function_domain, function_range, affine)

Bases: object

Class for affine transformation from domain to a range

This class has an affine attribute, which is a matrix representing the affine transformation in homogeneous coordinates. This matrix is used to evaluate the function, rather than having an explicit function (as is the case for a CoordinateMap).

Examples

```python
>>> inp_cs = CoordinateSystem('ijk')
>>> out_cs = CoordinateSystem('xyz')
>>> cm = AffineTransform(inp_cs, out_cs, np.diag([1, 2, 3, 1]))
>>> cm
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k'), name='', coord_dtype=float64),
```

538 Chapter 73. core.reference.coordinate_map
```python
function_range=CoordinateSystem(coord_names=('x', 'y', 'z'), name='', coord_dtype=float64),
affine=array([[ 1., 0., 0., 0.],
              [ 0., 2., 0., 0.],
              [ 0., 0., 3., 0.],
              [ 0., 0., 0., 1.]]))

>>> cm.affine
array([[ 1., 0., 0., 0.],
       [ 0., 2., 0., 0.],
       [ 0., 0., 3., 0.],
       [ 0., 0., 0., 1.]])

>>> cm([1,1,1])
array([ 1., 2., 3.])

>>> icm = cm.inverse()

>>> icm([1,2,3])
array([ 1., 1., 1.])
```

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__call__</code>(x)</td>
<td>Return mapping evaluated at x</td>
</tr>
<tr>
<td><code>from_params</code>(innames, outnames, params[, ...])</td>
<td>Create <code>AffineTransform</code> from <code>innames</code> and <code>outnames</code></td>
</tr>
<tr>
<td><code>from_start_step</code>(innames, outnames, start, step)</td>
<td>New <code>AffineTransform</code> from names, start and step.</td>
</tr>
<tr>
<td><code>identity</code>(coord_names[, name])</td>
<td>Return an identity coordmap of the given shape</td>
</tr>
<tr>
<td><code>inverse</code>(preserve_dtype)</td>
<td>Return coordinate map with inverse affine transform or None</td>
</tr>
<tr>
<td><code>renamed_domain</code>(newnames[, name])</td>
<td>New <code>AffineTransform</code> with function_domain renamed</td>
</tr>
<tr>
<td><code>renamed_range</code>(newnames[, name])</td>
<td>New <code>AffineTransform</code> with renamed function_domain</td>
</tr>
<tr>
<td><code>reordered_domain</code>([order])</td>
<td>New <code>AffineTransform</code> with function_domain reordered</td>
</tr>
<tr>
<td><code>reordered_range</code>([order])</td>
<td>New <code>AffineTransform</code> with function_range reordered</td>
</tr>
<tr>
<td><code>similar_to</code>(other)</td>
<td>Does <code>other</code> have similar coordinate systems and same mappings?</td>
</tr>
</tbody>
</table>

```python
__init__`(function_domain, function_range, affine)

Initialize `AffineTransform`

**Parameters**

- **function_domain**: `CoordinateSystem`
  input coordinates
- **function_range**: `CoordinateSystem`
  output coordinates
- **affine**: array-like
  affine homogenous coordinate matrix

**Notes**

The dtype of the resulting matrix is determined by finding a safe typecast for the function_domain, function_range and affine.

```python
affine = array([[3, 0, 0, 0], [0, 4, 0, 0], [0, 0, 5, 0], [0, 0, 0, 1]])
```

```
static from_params (innames, outnames, params, domain_name='', range_name='')
Create `AffineTransform` from `innames` and `outnames`
Parameters  

**innames**: sequence of str or str

The names of the axes of the domain. If str, then names given by list(innames)

**outnames**: sequence of str or str

The names of the axes of the range. If str, then names given by list(outnames)

**params**: AffineTransform, array or (array, array)

An affine function between the domain and range. This can be represented either by a single ndarray (which is interpreted as the representation of the function in homogeneous coordinates) or an (A,b) tuple.

**domain_name**: str, optional

Name of domain CoordinateSystem

**range_name**: str, optional

Name of range CoordinateSystem

Returns  **aff**: AffineTransform

Notes

**Precondition**  len(shape) == len(names)

**Raises** ValueError if len(shape) != len(names)

**static from_start_step** *(innames, outnames, start, step, domain_name='', range_name='')*

New AffineTransform from names, start and step.

Parameters  

**innames**: sequence of str or str

The names of the axes of the domain. If str, then names given by list(innames)

**outnames**: sequence of str or str

The names of the axes of the range. If str, then names given by list(outnames)

**start**: sequence of float

Start vector used in constructing affine transformation

**step**: sequence of float

Step vector used in constructing affine transformation

**domain_name**: str, optional

Name of domain CoordinateSystem

**range_name**: str, optional

Name of range CoordinateSystem

Returns  **cm**: CoordinateMap

Notes

len(names) == len(start) == len(step)
Examples

```python
>>> cm = AffineTransform.from_start_step('ijk', 'xyz', [1, 2, 3], [4, 5, 6])
>>> cm.affine
array([[4., 0., 0., 1.],
       [0., 5., 0., 2.],
       [0., 0., 6., 3.],
       [0., 0., 0., 1.]])
```

function_domain = CoordinateSystem(coord_names=('x',), name='', coord_dtype=float64)

function_range = CoordinateSystem(coord_names=('y',), name='', coord_dtype=float64)

static identity(coord_names, name='')
Return an identity coordmap of the given shape

Parameters  coord_names : sequence of str or str
   The names of the axes of the domain. If str, then names given by
   list(coord_names)

name : str, optional
   Name of origin of coordinate system

Returns  cm : CoordinateMap
   CoordinateMap with CoordinateSystem domain and an identity transform,
   with identical domain and range.

Examples

```python
>>> cm = AffineTransform.identity('ijk', 'somewhere')
>>> cm.affine
array([[1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
>>> cm.function_domain
CoordinateSystem(coord_names=('i', 'j', 'k'), name='somewhere', coord_dtype=float64)
>>> cm.function_range
CoordinateSystem(coord_names=('i', 'j', 'k'), name='somewhere', coord_dtype=float64)
```

inverse(preserve_dtype=False)
Return coordinate map with inverse affine transform or None

Parameters  preserve_dtype : bool
   If False, return affine mapping from inverting the affine. The domain / range dtypes
   for the inverse may then change as a function of the dtype of the inverted affine. If
   True, try to invert our affine, and see if it can be cast to the needed data type, which
   is self.function_domain.coord_dtype. We need this dtype in order for the
   inverse to preserve the coordinate system dtypes.

Returns  aff_cm_inv : AffineTransform instance or None
   AffineTransform mapping from the range of input self to the domain of input self -
   the inverse of self. If self.affine was not invertible return None. If
   preserve_dtype is True, and the inverse of self.affine cannot be cast to
   self.function_domain.coord_dtype, then return None. Otherwise return
AffineTransform inverse mapping. If `preserve_dtype` is False, the domain / range dtypes of the return inverse may well be different from those of the input `self`.

Examples

```python
def matrix_inverse(matrix):
    # Example of matrix inverse calculation
    inverse_matrix = np.linalg.inv(matrix)
    return inverse_matrix
```

The inverse transform, by default, generates a floating point inverse matrix and therefore floating point output:

```python
>>> affine_transform_inv = affine_transform.inverse()
>>> affine_transform_inv([2, 6, 12])
array([ 1., 5., 11.])
```

You can force it to preserve the coordinate system dtype with the `preserve_dtype` flag:

```python
>>> at_inv_preserved = affine_transform.inverse(preserve_dtype=True)
>>> at_inv_preserved([2, 6, 12])
array([ 1, 5, 11])
```

If you `preserve_dtype`, and there is no inverse affine preserving the dtype, the inverse is None:

```python
>>> affine2 = affine.copy()
>>> affine2[0, 0] = 2 # now inverse can't be integer
>>> aff_t = AffineTransform(input_cs, output_cs, affine2)
>>> aff_t.inverse(preserve_dtype=True) is None
True
```

ndims = (3, 3)

renamed_domain (newnames, name='')
New AffineTransform with function_domain renamed

Parameters newnames : dict
A dictionary whose keys are integers or are in mapping.function_domain.coord_names and whose values are the new names.

Returns newmapping : AffineTransform
A new AffineTransform with renamed function_domain.

Examples

```python
>>> affine_domain = CoordinateSystem('ijk')
>>> affine_range = CoordinateSystem('xyz')
>>> affine_matrix = np.identity(4)
>>> affine_mapping = AffineTransform(affine_domain, affine_range, affine_matrix)
```
>>> new_affine_mapping = affine_mapping.renamed_domain({'i':'phase', 'k':'freq', 'j':'slice'})
>>> new_affine_mapping.function_domain
CoordinateSystem(coord_names=('phase', 'slice', 'freq'), name='', coord_dtype=float64)

>>> new_affine_mapping = affine_mapping.renamed_domain({'i':'phase', 'k':'freq', 'l':'slice'})
Traceback (most recent call last):
  ...          
ValueError: no domain coordinate named l

renamed_range(newnames, name='')
New AffineTransform with renamed function_domain

Parameters newnames : dict
A dictionary whose keys are integers or are in mapping.function_range.coord_names
and whose values are the new names.

Returns newmapping : AffineTransform
A new AffineTransform with renamed function_range.

Examples

>>> affine_domain = CoordinateSystem('ijk')
>>> affine_range = CoordinateSystem('xyz')
>>> affine_matrix = np.identity(4)
>>> affine_mapping = AffineTransform(affine_domain, affine_range, affine_matrix)

>>> new_affine_mapping = affine_mapping.renamed_range({'x':'u'})
>>> new_affine_mapping.renamed_range({'w':'u'})
Traceback (most recent call last):
  ...          
ValueError: no range coordinate named w

reordered_domain(order=None)
New AffineTransform with function_domain reordered

Default behaviour is to reverse the order of the coordinates.

Parameters order : sequence
Order to use, defaults to reverse. The elements can be integers, strings or 2-tuples of
strings. If they are strings, they should be in mapping.function_domain.coord_names.

Returns newmapping : AffineTransform
A new AffineTransform with the coordinates of function_domain reordered.

Examples

>>> input_cs = CoordinateSystem('ijk')
>>> output_cs = CoordinateSystem('xyz')
>>> cm = AffineTransform(input_cs, output_cs, np.identity(4))
>>> cm.reordered_domain('ikj').function_domain
CoordinateSystem(coord_names=('i', 'k', 'j'), name='', coord_dtype=float64)
reordered_range(order=None)
  New AffineTransform with function_range reordered
  Defaults to reversing the coordinates of function_range.

  Parameters order : sequence
    Order to use, defaults to reverse. The elements can be integers, strings or 2-tuples of
    strings. If they are strings, they should be in mapping.function_range.coord_names.

  Returns newmapping : AffineTransform
    A new AffineTransform with the coordinates of function_range reordered.

Examples

>>> input_cs = CoordinateSystem('ijk')
>>> output_cs = CoordinateSystem('xyz')
>>> cm = AffineTransform(input_cs, output_cs, np.identity(4))
>>> cm.reordered_range('xzy').function_range
CoordinateSystem(coord_names=('x', 'z', 'y'), name='', coord_dtype=float64)
>>> cm.reordered_range([0,2,1]).function_range.coord_names
('x', 'z', 'y')

>>> newcm = cm.reordered_range('yzx')
>>> newcm.function_range.coord_names
('y', 'z', 'x')

similar_to(other)
  Does other have similar coordinate systems and same mappings?
  A "similar" coordinate system is one with the same coordinate names and data dtype, but ignoring the
  coordinate system name.

73.2.2 AxisError

class nipy.core.reference.coordinate_map.AxisError
  Bases: exceptions.Exception
  Error for incorrect axis selection

  __init__(self, args, message)
    x.__init__(...) initializes x; see help(type(x)) for signature

    args
    message

73.2.3 CoordMapMaker

class nipy.core.reference.coordinate_map.CoordMapMaker(domain_maker, range_maker)
  Bases: object
  Class to create coordinate maps of different dimensions

    Methods
__call__(*args, **kwargs) Create affine or non-affine coordinate map

affine_maker

generic_maker

make_affine(affine[, append_zooms, ...]) Create affine coordinate map

make_cmap(domain_N, xform[, inv_xform]) Coordinate map with transform function xform

__init__ (domain_maker, range_maker)
Create coordinate map maker

Parameters  
  domain_maker : callable
      A coordinate system maker, returning a coordinate system with input argument only N, 
an integer giving the length of the coordinate map.

  range_maker : callable
      A coordinate system maker, returning a coordinate system with input argument only N, 
an integer giving the length of the coordinate map.

Examples

>>> from nipy.core.reference.coordinate_system import CoordSysMaker
>>> dmaker = CoordSysMaker('ijkl', 'generic-array')
>>> rmaker = CoordSysMaker('xyzt', 'generic-scanner')
>>> cm_maker = CoordMapMaker(dmaker, rmaker)

affine_maker
alias of AffineTransform

generic_maker
alias of CoordinateMap

make_affine(affine, append_zooms=(), append_offsets=())
Create affine coordinate map

Parameters  
  affine : (M, N) array-like
      Array expressing the affine tranformation

  append_zooms : scalar or sequence length E
      If scalar, converted to sequence length E==1. Append E entries to the diagonal of affine 
      (see examples)

  append_offsets : scalar or sequence length F
      If scalar, converted to sequence length F==1. If F==0, and E!=0, use sequence of zeros 
      length E. Append E entries to the translations (final column) of affine (see examples).

Returns  
  affmap : AffineTransform coordinate map

Examples

>>> from nipy.core.reference.coordinate_system import CoordSysMaker
>>> dmaker = CoordSysMaker('ijkl', 'generic-array')
>>> rmaker = CoordSysMaker('xyzt', 'generic-scanner')
>>> cm_maker = CoordMapMaker(dmaker, rmaker)
>>> cm_maker.make_affine(np.diag([2, 3, 4, 1]))
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k'), name='generic-array', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('x', 'y', 'z'), name='generic-scanner', coord_dtype=float64),
    affine=array([[ 2., 0., 0., 0.],
                   [ 0., 3., 0., 0.],
                   [ 0., 0., 4., 0.],
                   [ 0., 0., 0., 1.]]))

We can add extra orthogonal dimensions, by specifying the diagonal elements:

>>> cm_maker.make_affine(np.diag([2, 3, 4, 1]), 6)
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='generic-array', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('x', 'y', 'z', 't'), name='generic-scanner', coord_dtype=float64),
    affine=array([[ 2., 0., 0., 0., 0.],
                   [ 0., 3., 0., 0., 0.],
                   [ 0., 0., 4., 0., 0.],
                   [ 0., 0., 0., 6., 0.],
                   [ 0., 0., 0., 0., 1.]]))

Or the diagonal elements and the offset elements:

>>> cm_maker.make_affine(np.diag([2, 3, 4, 1]), [6], [9])
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='generic-array', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('x', 'y', 'z', 't'), name='generic-scanner', coord_dtype=float64),
    affine=array([[ 2., 0., 0., 0., 0.],
                   [ 0., 3., 0., 0., 0.],
                   [ 0., 0., 4., 0., 0.],
                   [ 0., 0., 0., 6., 9.],
                   [ 0., 0., 0., 0., 1.]]))

make_cmap (domain_N, xform, inv_xform=None)
Coordinate map with transform function xform

Parameters  

- domain_N : int
  Number of domain coordinates

- xform : callable
  Function that transforms points of dimension domain_N

- inv_xform : None or callable, optional
  Function, such that inv_xform(xform(pts)) returns pts

Returns  cmap : CoordinateMap

Examples

>>> from nipy.core.reference.coordinate_system import CoordSysMaker
>>> dmaker = CoordSysMaker('ijkl', 'generic-array')
>>> rmaker = CoordSysMaker('xyzt', 'generic-scanner')
>>> cm_maker = CoordMapMaker(dmaker, rmaker)
>>> cm_maker.make_cmap(4, lambda x : x+1)
73.2.4 CoordMapMakerError

class nipy.core.reference.coordinate_map.CordMapMakerError
Bases: exceptions.Exception

__init__()
    x.__init__(...) initializes x; see help(type(x)) for signature

args

message

73.2.5 CoordinateMap

class nipy.core.reference.coordinate_map.CordinateMap (function_domain, function_range, function, inverse_function=None)

Bases: object

A set of domain and range CoordinateSystems and a function between them.

For example, the function may represent the mapping of a voxel (the domain of the function) to real space (the range). The function may be an affine or non-affine transformation.

Examples

>>> function_domain = CoordinateSystem(‘ijk’, ‘voxels’)
>>> function_range = CoordinateSystem(‘xyz’, ‘world’)
>>> mni_orig = np.array([-90.0, -126.0, -72.0])
>>> function = lambda x: x + mni_orig
>>> inv_function = lambda x: x - mni_orig
>>> cm = CoordinateMap(function_domain, function_range, function, inv_function)

Map the first 3 voxel coordinates, along the x-axis, to mni space:

>>> x = np.array([[0,0,0], [1,0,0], [2,0,0]])
>>> cm.function(x)
array([[ -90., -126., -72.],
       [ -89., -126., -72.],
       [ -88., -126., -72.]])

>>> x = CoordinateSystem(‘x’)
>>> y = CoordinateSystem(‘y’)
>>> m = CoordinateMap(x, y, np.exp, np.log)
>>> m
CoordinateMap(  
    function_domain=CoordinateSystem(coord_names=('x',), name='', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('y',), name='', coord_dtype=float64),
    function=<ufunc 'exp'>,
    inverse_function=<ufunc 'log'>
)
m.inverse()

CoordinateMap(
    function_domain=CoordinateSystem(coord_names=('y',), name='', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('x',), name='', coord_dtype=float64),
    function=<ufunc 'log'>,
    inverse_function=<ufunc 'exp'>
)

Attributes

- **function_domain**: An ordered sequence of named coordinates of a specified dtype.
- **function_range**: An ordered sequence of named coordinates of a specified dtype.
- **function**: An ordered sequence of named coordinates of a specified dtype.
- **inverse_function**: Natural logarithm, element-wise.

Methods

- __call__(x) - Return mapping evaluated at x
- function(x[, out]) - Calculate the exponential of all elements in the input array.
- inverse() - New CoordinateMap with the functions reversed
- inverse_function(x[, out]) - Natural logarithm, element-wise.
- renamed_domain(newnames[, name]) - New CoordinateMap with function_domain renamed
- renamed_range(newnames[, name]) - New CoordinateMap with function_range renamed
- reordered_domain([order]) - Create a new CoordinateMap with the coordinates of function_domain reordered.
- reordered_range([order]) - New CoordinateMap with function_range reordered.
- similar_to(other) - Does other have similar coordinate systems and same mappings?

___init__(function_domain, function_range, function, inverse_function=None)

Create a CoordinateMap given function, domain and range.

**Parameters**

- **function_domain**: CoordinateSystem
  - The input coordinate system.
- **function_range**: CoordinateSystem
  - The output coordinate system
- **function**: callable
  - The function between function_domain and function_range. It should be a callable that accepts arrays of shape (N, function_domain.ndim) and returns arrays of shape (N, function_range.ndim), where N is the number of points for transformation.
- **inverse_function**: None or callable, optional
  - The optional inverse of function, with the intention being \( x = inverse_function(function(x)) \). If the function is affine and invertible, then this is true for all x. The default is None

**Returns**

- **coordmap**: CoordinateMap
  - \( function(x[, out]) = <ufunc 'exp'> \)
function_domain = CoordinateSystem(coord_names=('x',), name='', coord_dtype=float64)
function_range = CoordinateSystem(coord_names=('y',), name='', coord_dtype=float64)

inverse()
    New CoordinateMap with the functions reversed

inverse_function (x[, out]) = <ufunc 'log'>

ndims = (1, 1)

renamed_domain (newnames, name='')
    New CoordinateMap with function_domain renamed

Parameters

    newnames : dict
A dictionary whose keys are integers or are in mapping.function_domain.coord_names
and whose values are the new names.

Returns

    newmapping : CoordinateMap
A new CoordinateMap with renamed function_domain.

Examples

>>> domain = CoordinateSystem('ijk')
>>> range = CoordinateSystem('xyz')
>>> cm = CoordinateMap(domain, range, lambda x:x+1)

>>> new_cm = cm.renamed_domain({'i':'phase','k':'freq','j':'slice'})
>>> new_cm.function_domain
CoordinateSystem(coord_names=('phase', 'slice', 'freq'), name='', coord_dtype=float64)

>>> new_cm = cm.renamed_domain({'i':'phase','k':'freq','l':'slice'})
Traceback (most recent call last):
  ...
    ValueError: no domain coordinate named 1

renamed_range (newnames, name='')
    New CoordinateMap with function_domain renamed

Parameters

    newnames : dict
A dictionary whose keys are integers or are in mapping.function_range.coord_names
and whose values are the new names.

Returns

    newmapping : CoordinateMap
A new CoordinateMap with renamed function_range.

Examples

>>> domain = CoordinateSystem('ijk')
>>> range = CoordinateSystem('xyz')
>>> cm = CoordinateMap(domain, range, lambda x:x+1)

>>> new_cm = cm.renamed_range({'x':'u'})
>>> new_cm.function_range
CoordinateSystem(coord_names=('u', 'y', 'z'), name='', coord_dtype=float64)
new_cm = cm.renamed_range({'w':'u'})
ValueError: no range coordinate named w

reordered_domain(order=None)
Create a new CoordinateMap with the coordinates of function_domain reordered. Default behaviour is to reverse the order of the coordinates.

Parameters order : sequence
Order to use, defaults to reverse. The elements can be integers, strings or 2-tuples of strings. If they are strings, they should be in mapping.function_domain.coord_names.

Returns newmapping : CoordinateMap
A new CoordinateMap with the coordinates of function_domain reordered.

Examples

>>> input_cs = CoordinateSystem('ijk')
>>> output_cs = CoordinateSystem('xyz')
>>> cm = CoordinateMap(input_cs, output_cs, lambda x:x+1)
>>> cm.reordered_domain('ikj').function_domain
CoordinateSystem(coord_names=('i', 'k', 'j'), name='', coord_dtype=float64)

reordered_range(order=None)
New CoordinateMap with function_range reordered.

Defaults to reversing the coordinates of function_range.

Parameters order : sequence
Order to use, defaults to reverse. The elements can be integers, strings or 2-tuples of strings. If they are strings, they should be in mapping.function_range.coord_names.

Returns newmapping : CoordinateMap
A new CoordinateMap with the coordinates of function_range reordered.

Examples

>>> input_cs = CoordinateSystem('ijk')
>>> output_cs = CoordinateSystem('xyz')
>>> cm = CoordinateMap(input_cs, output_cs, lambda x:x+1)
>>> cm.reordered_range('xzy').function_range
CoordinateSystem(coord_names=('x', 'z', 'y'), name='', coord_dtype=float64)
>>> cm.reordered_range([0,2,1]).function_range.coord_names
('x', 'z', 'y')
>>> newcm = cm.reordered_range('yzx')
>>> newcm.function_range.coord_names
('y', 'z', 'x')

similar_to(other)
Does other have similar coordinate systems and same mappings?

A "similar" coordinate system is one with the same coordinate names and data dtype, but ignoring the coordinate system name.
73.3 Functions

nipy.core.reference.coordinate_map.append_io_dim(cm, in_name, out_name, start=0, step=1)

Append input and output dimension to coordmap

Parameters cm : Affine

Affine coordinate map instance to which to append dimension

in_name : str

Name for new input dimension

out_name : str

Name for new output dimension

start : float, optional

Offset for transformed values in new dimension

step : float, optional

Step, or scale factor for transformed values in new dimension

Returns cm_plus : Affine

New coordinate map with appended dimension

Examples

Typical use is creating a 4D coordinate map from a 3D

```python
>>> cm3d = AffineTransform.from_params('ijk', 'xyz', np.diag([1,2,3,1]))
>>> cm4d = append_io_dim(cm3d, 'l', 't', 9, 5)
>>> cm4d.affine
array([[ 1., 0., 0., 0., 0.],
[ 0., 2., 0., 0., 0.],
[ 0., 0., 3., 0., 0.],
[ 0., 0., 0., 5., 9.],
[ 0., 0., 0., 0., 1.]])
```

nipy.core.reference.coordinate_map.axmap(coordmap, direction='in2out', fix0=True)

Return mapping between input and output axes

Parameters coordmap : Affine

Affine coordinate map instance for which to get axis mappings

direction : {'in2out', 'out2in', 'both'}

direction to find mapping. If 'in2out', returned mapping will have keys from the input axis (names and indices) and values of corresponding output axes. If 'out2in' the keys will be output axis names, indices and the values will be input axis indices. If both, return both mappings.

fix0 : bool, optional

Whether to fix potential 0 TR in affine

Returns map : dict or tuple
• if `direction` == ‘in2out’ - mapping with keys of input names and input indices, values of output indices. Mapping is to closest matching axis. None means there appears to be no matching axis

• if `direction` == ‘out2in’ - mapping with keys of output names and input indices, values of input indices, as above.

• if `direction` == ‘both’ - tuple of (input to output mapping, output to input mapping)

```python
nipy.core.reference.coordinate_map.compose(*cmaps)
```

Return the composition of two or more CoordinateMaps.

**Parameters**
- **cmaps**: sequence of CoordinateMaps

**Returns**
- **cmap**: CoordinateMap
  - The resulting CoordinateMap has `function_domain == cmaps[-1].function_domain` and `function_range == cmaps[0].function_range`

**Examples**

```python
cmap = AffineTransform.from_params('i', 'x', np.diag([2.,1.]))
cmapi = cmap.inverse()
id1 = compose(cmap,cmapi)
id1.affine
array([[ 1., 0.],
        [ 0., 1.]]

id2 = compose(cmapi,cmap)
id1.function_domain.coord_names
('x',)
id2.function_domain.coord_names
('i',)
```

```python
nipy.core.reference.coordinate_map.drop_io_dim(cm, axis_id, fix0=True)
```

Drop dimensions `axis_id` from coordinate map, if orthogonal to others

If you specify an input dimension, drop that dimension and any corresponding output dimension, as long as all other outputs are orthogonal to dropped input. If you specify an output dimension, drop that dimension and any corresponding input dimension, as long as all other inputs are orthogonal to dropped output.

**Parameters**
- **cm**: class:AffineTransform
  - Affine coordinate map instance
- **axis_id**: int or str
  - If int, gives index of `input` axis to drop. If str, gives name of `input` or `output` axis to drop. When specifying an input axis: if given input axis does not affect any output axes, just drop input axis. If input axis affects only one output axis, drop both input and corresponding output. Similarly when specifying an output axis. If `axis_id` is a str, it must be unambiguous - if the named axis exists in both input and output, and they do not correspond, raises a AxisError. See Raises section for checks
- **fix0**: bool, optional
  - Whether to fix potential 0 TR in affine

**Returns**
- **cm_redux**: Affine
  - Affine coordinate map with orthogonal input + output dimension dropped
Raises AxisError: if `axis_id` is a str and does not match any no input or output:
    coordinate names.

AxisError: if specified `axis_id` affects more than a single input / output:
    axis.

AxisError: if the named `axis_id` exists in both input and output, and they:
    do not correspond.

Examples

Typical use is in getting a 3D coordinate map from 4D

```python
>>> cm4d = AffineTransform.from_params('ijkl', 'xyzt', np.diag([1,2,3,4,1]))
>>> cm3d = drop_io_dim(cm4d, 't')
>>> cm3d.affine
array([[ 1.,  0.,  0.,  0.],
        [ 0.,  2.,  0.,  0.],
        [ 0.,  0.,  3.,  0.],
        [ 0.,  0.,  0.,  1.]])
```

nipy.core.reference.coordinate_map.equivalent(mapping1, mapping2)

A test to see if mapping1 is equal to mapping2 after possibly reordering the domain and range of mapping.

**Parameters**

- **mapping1**: CoordinateMap or AffineTransform
- **mapping2**: CoordinateMap or AffineTransform

**Returns**

- **are_they_equal**: bool

Examples

```python
>>> ijk = CoordinateSystem('ijk')
>>> xyz = CoordinateSystem('xyz')
>>> T = np.random.standard_normal((4,4))
>>> T[-1] = [0,0,0,1]  # otherwise AffineTransform raises
...                     # an exception because
...                     # it’s supposed to represent an
...                     # affine transform in homogeneous
...                     # coordinates
>>> A = AffineTransform(ijk, xyz, T)
>>> B = A.reordered_domain('ikj').reordered_range('xzy')
>>> C = B.renamed_domain({'i':'slice'})
>>> equivalent(A, B)
True
>>> equivalent(A, C)
False
>>> equivalent(B, C)
False
>>> D = CoordinateMap(ijk, xyz, np.exp)
>>> equivalent(D, D)
True
>>> E = D.reordered_domain('kij').reordered_range('xzy')
>>> # no non-AffineTransform will ever be
>>> # equivalent to a reordered version of itself,
```
(97x696)because their functions don’t evaluate as equal
>>> equivalent(D, E)
False
>>> equivalent(E, E)
True
>>> # This has not changed the order
>>> # of the axes, so the function is still the same
>>> F = D.reordered_range('xyz').reordered_domain('ijk')
>>> equivalent(F, D)
True
>>> id(F) == id(D)
False

nipy.core.reference.coordinate_map.input_axis_index(coordmap, axis_id, fix0=True)

Return input axis index for `axis_id`

`axis_id` can be integer, or a name of an input axis, or it can be the name of an output axis which maps to an input axis.

Parameters

**coordmap**: AffineTransform

**axis_id**: int or str

If int, then an index of an input axis. Can be negative, so that -2 refers to the second to last input axis. If a str can be the name of an input axis, or the name of an output axis that should have a corresponding input axis (see Raises section).

**fix0**: bool, optional

Whether to fix potential single 0 on diagonal of affine. This often happens when loading nifti images with TR set to 0.

Returns

**inax**: int

index of matching input axis. If `axis_id` is the name of an output axis, then `inax` will be the input axis that had a ‘best’ match with this output axis. The ‘best’ match algorithm ensures that there can only be one input axis paired with one output axis.

Raises

**AxisError**: if no matching name found

**AxisError**: if name exists in both input and output, and they do not map to each other

**AxisError**: if name present in output but no matching input

nipy.core.reference.coordinate_map.io_axis_indices(coordmap, axis_id, fix0=True)

Return input and output axis index for id `axis_id` in `coordmap`

Parameters

**cm**: class:AffineTransform

Affine coordinate map instance

**axis_id**: int or str

If int, gives index of input axis. Can be negative, so that -2 refers to the second from last input axis. If str, gives name of input or output axis. If `axis_id` is a str, it must be unambiguous - if the named axis exists in both input and output, and they do not correspond, raises a AxisError. See Raises section for checks.

**fix0**: bool, optional
Whether to fix potential 0 column / row in affine

**Returns**  
in\_index : None or int  
index of input axis that corresponds to axis\_id  
out\_index : None or int  
index of output axis that corresponds to axis\_id

**Raises**  
AxisError: if ‘axis\_id’ is a str and does not match any input or output coordinate names.

AxisError: if the named ‘axis\_id’ exists in both input and output, and they do not correspond.

**Examples**

```python
g = [[0, 1, 0, 10], [1, 0, 0, 11], [0, 0, 1, 12], [0, 0, 0, 1]]  
gmap = AffineTransform('ijk', 'xyz', g)  
io_axis_indices(gmap, 0)  
(0, 1)  
io_axis_indices(gmap, 1)  
(1, 0)  
io_axis_indices(gmap, -1)  
(2, 2)  
io_axis_indices(gmap, 'j')  
(1, 0)  
io_axis_indices(gmap, 'y')  
(0, 1)
```

nipy.core.reference.coordinate_map.orth\_axes(in\_ax, out\_ax, affine, allow\_zero=True, tol=1e-05)

True if in\_ax related only to out\_ax in affine and vice versa

**Parameters**  
in\_ax : int  
Input axis index  
out\_ax : int  
Output axis index  
affine : array-like  
Affine transformation matrix  
allow\_zero : bool, optional  
Whether to allow zero in affine[out\_ax, in\_ax]. This means that the two axes are not related, but nor is this pair related to any other part of the affine.

**Returns**  
tf : bool  
True if in\_ax, out\_ax pair are orthogonal to the rest of affine, unless allow\_zero is False, in which case require in addition that affine[out\_ax, in\_ax] != 0.
Examples

```python
>>> aff = np.eye(4)
>>> orth_axes(1, 1, aff)
True
>>> orth_axes(1, 2, aff)
False
```

```
nipy.core.reference.coordinate_map.product(*cmaps, **kwargs)

“topological” product of two or more mappings

The mappings can be either AffineTransforms or CoordinateMaps.

If they are all AffineTransforms, the result is an AffineTransform, else it is a CoordinateMap.

Parameters  cmaps : sequence of CoordinateMaps or AffineTransforms

Returns  cmap : CoordinateMap
```

Examples

```python
>>> inc1 = AffineTransform.from_params('i', 'x', np.diag([2,1]))
>>> inc2 = AffineTransform.from_params('j', 'y', np.diag([3,1]))
>>> inc3 = AffineTransform.from_params('k', 'z', np.diag([4,1]))

>>> cmap = product(inc1, inc3, inc2)
>>> cmap.function_domain.coord_names
('i', 'k', 'j')
>>> cmap.function_range.coord_names
('x', 'z', 'y')
>>> cmap.affine
array([[ 2., 0., 0., 0.],
        [ 0., 4., 0., 0.],
        [ 0., 0., 3., 0.],
        [ 0., 0., 0., 1.]])

>>> A1 = AffineTransform.from_params('ij', 'xyz', np.array([[2,3,1,0],[3,4,5,0],[7,9,3,1]]).T)
>>> A2 = AffineTransform.from_params('xyz', 'de', np.array([[8,6,7,4],[1,-1,13,3],[0,0,0,1]]))

>>> A1.affine
array([[ 2., 3., 7.],
        [ 3., 4., 9.],
        [ 1., 5., 3.],
        [ 0., 0., 1.]])
>>> A2.affine
array([[ 8., 6., 7., 4.],
        [ 1., -1., 13., 3.],
        [ 0., 0., 0., 1.]])

>>> p=product(A1, A2)
>>> p.affine
array([[ 2., 3., 0., 0., 0., 7.],
        [ 3., 4., 0., 0., 0., 9.],
        [ 1., 5., 0., 0., 0., 3.],
        [ 0., 0., 8., 6., 7., 4.],
        [ 0., 0., 1., -1., 13., 3.],
        [ 0., 0., 0., 0., 0., 1.]])
```
>>> np.allclose(p.affine[:3,:2], A1.affine[:3,:2])
True
>>> np.allclose(p.affine[:3,-1], A1.affine[:3,-1])
True
>>> np.allclose(p.affine[3:5,2:5], A2.affine[:2,:3])
True
>>> np.allclose(p.affine[3:5,-1], A2.affine[:2,-1])
True

```python
>>> A1([3,4])
aarray([ 25., 34., 26.])
>>> A2([5,6,7])
aarray([ 129., 93.])
>>> p([3,4,5,6,7])
aarray([ 25., 34., 26., 129., 93.])
```

nipy.core.reference.coordinate_map.\texttt{renamed\_domain}(mapping, \texttt{newnames}, \texttt{name=None})

New coordmap with the coordinates of function\_domain renamed

**Parameters**  
\texttt{newnames}: dict  
A dictionary whose keys are integers or are in mapping.function\_range.coord\_names and whose values are the new names.

**Returns**  
\texttt{newmapping}: CoordinateMap or AffineTransform  
A new mapping with renamed function\_domain. If isinstance(mapping, AffineTransform), newmapping is also an AffineTransform. Otherwise, it is a CoordinateMap.

**Examples**

```python
>>> affine\_domain = CoordinateSystem(‘ijk’)  
>>> affine\_range = CoordinateSystem(‘xyz’)  
>>> affine\_matrix = np.identity(4)  
>>> affine\_mapping = AffineTransform(affine\_domain, affine\_range, affine\_matrix)

>>> new\_affine\_mapping = affine\_mapping.\texttt{renamed\_domain}(\{‘i’:‘phase’, ‘k’:‘freq’, ‘j’:‘slice’\})
>>> new\_affine\_mapping.function\_domain
CoordinateSystem(coord\_names=(‘phase’, ‘slice’, ‘freq’), name=’, coord\_dtype=float64)

>>> new\_affine\_mapping = affine\_mapping.\texttt{renamed\_domain}(\{‘i’:‘phase’, ‘k’:‘freq’, ‘l’:‘slice’\})
Traceback (most recent call last):
...  
ValueError: no domain coordinate named l
```

nipy.core.reference.coordinate_map.\texttt{renamed\_range}(mapping, \texttt{newnames})

New coordmap with the coordinates of function\_range renamed

**Parameters**  
\texttt{newnames}: dict  
A dictionary whose keys are integers or in mapping.function\_range.coord\_names and whose values are the new names.

**Returns**  
\texttt{newmapping}: CoordinateMap or AffineTransform  
A new CoordinateMap with the coordinates of function\_range renamed. If isinstance(mapping, AffineTransform), newmapping is also an AffineTransform. Otherwise, it is a CoordinateMap.
Examples

>>> affine_domain = CoordinateSystem('ijk')
>>> affine_range = CoordinateSystem('xyz')
>>> affine_matrix = np.identity(4)
>>> affine_mapping = AffineTransform(affine_domain, affine_range, affine_matrix)
>>> new_affine_mapping = affine_mapping.renamed_range({'x':'u'})
>>> new_affine_mapping.function_range
CoordinateSystem(coord_names=('u', 'y', 'z'), name='', coord_dtype=float64)

>>> new_affine_mapping = affine_mapping.renamed_range({'w':'u'})
Traceback (most recent call last):
...
ValueError: no range coordinate named w

nipy.core.reference.coordinate_map.reordered_domain(mapping, order=None)
New coordmap with the coordinates of function_domain reordered

Default behaviour is to reverse the order of the coordinates.

Parameters  order: sequence :

Order to use, defaults to reverse. The elements can be integers, strings or 2-tuples of
strings. If they are strings, they should be in mapping.function_domain.coord_names.

Returns  newmapping : CoordinateMap or AffineTransform

A new CoordinateMap with the coordinates of function_domain reordered. If isin-
stance(mapping, AffineTransform), newmapping is also an AffineTransform. Other-
wise, it is a CoordinateMap.

Notes

If no reordering is to be performed, it returns a copy of mapping.

Examples

>>> input_cs = CoordinateSystem('ijk')
>>> output_cs = CoordinateSystem('xyz')
>>> cm = AffineTransform(input_cs, output_cs, np.identity(4))
>>> cm.reordered_domain('ikj').function_domain
CoordinateSystem(coord_names=('i', 'k', 'j'), name='', coord_dtype=float64)

nipy.core.reference.coordinate_map.reordered_range(mapping, order=None)
New coordmap with the coordinates of function_range reordered

Defaults to reversing the coordinates of function_range.

Parameters  order: sequence :

Order to use, defaults to reverse. The elements can be integers, strings or 2-tuples of
strings. If they are strings, they should be in mapping.function_range.coord_names.

Returns  newmapping : CoordinateMap or AffineTransform

A new CoordinateMap with the coordinates of function_range reordered. If isin-
stance(mapping, AffineTransform), newmapping is also an AffineTransform. Other-
wise, it is a CoordinateMap.
Notes

If no reordering is to be performed, it returns a copy of mapping.

Examples

```python
>>> input_cs = CoordinateSystem('ijk')
>>> output_cs = CoordinateSystem('xyz')
>>> cm = AffineTransform(input_cs, output_cs, np.identity(4))
>>> cm.reordered_range('xzy').function_range
CoordinateSystem(coord_names=('x', 'z', 'y'), name='', coord_dtype=float64)
>>> cm.reordered_range([0,2,1]).function_range.coord_names
('x', 'z', 'y')

>>> newcm = cm.reordered_range('yzx')
>>> newcm.function_range.coord_names
('y', 'z', 'x')
```

nipy.core.reference.coordinate_map.shifted_domain_origin(mapping, difference_vector, new_origin)

Shift the origin of the domain

Parameters
difference_vector : array

Representing the difference shifted_origin-current_origin in the domain’s basis.

Examples

```python
>>> A = np.random.standard_normal((5, 6))
>>> A[-1] = [0,0,0,0,0,1]
>>> affine_transform = AffineTransform(CS('ijklm', 'oldorigin'), CS('xyzt'), A)
>>> affine_transform.function_domain
CoordinateSystem(coord_names=('i', 'j', 'k', 'l', 'm'), name='oldorigin', coord_dtype=float64)

A random change of origin

>>> difference = np.random.standard_normal(5)

The same affine transforation with a different origin for its domain

>>> shifted_affine_transform = shifted_domain_origin(affine_transform, difference, 'neworigin')
>>> shifted_affine_transform.function_domain
CoordinateSystem(coord_names=('i', 'j', 'k', 'l', 'm'), name='neworigin', coord_dtype=float64)

Let’s check that things work

>>> point_in_old_basis = np.random.standard_normal(5)

This is the relation ship between coordinates in old and new origins

>>> np.allclose(shifted_affine_transform(point_in_old_basis), affine_transform(point_in_old_basis+difference))
True
>>> np.allclose(shifted_affine_transform(point_in_old_basis-difference), affine_transform(point_in_old_basis))
True
```
nipy.core.reference.coordinate_map.shifted_range_origin(mapping, difference_vector, new_origin)

Shift the origin of the range.

Parameters  difference_vector : array

Representing the difference shifted_origin-current_origin in the range’s basis.

Examples

```python
>>> A = np.random.standard_normal((5,6))
>>> A[-1] = [0,0,0,0,0,1]
>>> affine_transform = AffineTransform(CS('ijklm'), CS('xymt', 'oldorigin'), A)
>>> affine_transform.function_range
CoordinateSystem(coord_names=('x', 'y', 'z', 't'), name='oldorigin', coord_dtype=float64)

Make a random shift of the origin in the range

>>> difference = np.random.standard_normal(4)
>>> shifted_affine_transform = shifted_range_origin(affine_transform, difference, 'neworigin')
>>> shifted_affine_transform.function_range
CoordinateSystem(coord_names=('x', 'y', 'z', 't'), name='neworigin', coord_dtype=float64)

Evaluate the transform and verify it does as expected

>>> point_in_domain = np.random.standard_normal(5)

Check that things work

>>> np.allclose(shifted_affine_transform(point_in_domain), affine_transform(point_in_domain) - difference)
True
>>> np.allclose(shifted_affine_transform(point_in_domain) + difference, affine_transform(point_in_domain))
True
```
CHAPTER SEVENTYFOUR

CORE.REFERENCE.COORDINATE_SYSTEM

74.1 Module: core.reference.coordinate_system

Inheritance diagram for nipy.core.reference.coordinate_system:

```
    reference.coordinate_system.CoordSysMaker
    reference.coordinate_system.CoordinateSystem
    reference.coordinate_system.CoordSysMakerError
    reference.coordinate_system.CoordinateSystemError
```

CoordinateSystems are used to represent the space in which the image resides.
A CoordinateSystem contains named coordinates, one for each dimension and a coordinate dtype. The purpose of the
CoordinateSystem is to specify the name and order of the coordinate axes for a particular space. This allows one to
compare two CoordinateSystems to determine if they are equal.

74.2 Classes

74.2.1 CoordSysMaker

class nipy.core.reference.coordinate_systemCoordSysMaker(coord_names, name='',
coord_dtype=<type
'float'>)

    Bases: object

    Class to create similar coordinate maps of different dimensions
Methods

```python
__call__(N[, name, coord_dtype])  # Create coordinate system of length N

coord_sys_klass

__init__(coord_names, name='', coord_dtype=<type 'float'>)
# Create a coordsys maker with given axis coord_names

Parameters
- coord_names: iterable
  A sequence of coordinate names.
- name: string, optional
  The name of the coordinate system
- coord_dtype: np.dtype, optional
  The dtype of the coord_names. This should be a built-in numpy scalar dtype. (default is np.float). The value can by anything that can be passed to the np.dtype constructor. For example np.float, np.dtype(np.float) or f8 all result in the same coord_dtype.

Examples

```python
>>> cmkr = CoordSysMaker('ijk', 'a name')
>>> print cmkr(2)
CoordinateSystem(coord_names=('i', 'j'), name='a name', coord_dtype=float64)
>>> print cmkr(3)
CoordinateSystem(coord_names=('i', 'j', 'k'), name='a name', coord_dtype=float64)
```

coord_sys_klass
# alias of CoordinateSystem
```

74.2.2 CoordSysMakerError

class nipy.core.reference.coordinate_system.CordSysMakerError
# Bases: exceptions.Exception

__init__() x.__init__(...) initializes x; see help(type(x)) for signature

args

message

74.2.3 CoordinateSystem

class nipy.core.reference.coordinate_system.CoordinateSystem(coord_names,
name='', coord_dtype=<type 'float'>)

Bases: object

An ordered sequence of named coordinates of a specified dtype.
A coordinate system is defined by the names of the coordinates, (attribute `coord_names`) and the numpy dtype of each coordinate value (attribute `coord_dtype`). The coordinate system can also have a name.

```python
>>> names = ['first', 'second', 'third']
>>> cs = CoordinateSystem(names, 'a coordinate system', np.float)
>>> cs.coord_names
('first', 'second', 'third')
>>> cs.name
'a coordinate system'
>>> cs.coord_dtype
dtype('float64')
```

The coordinate system also has a `dtype` which is the composite numpy dtype, made from the `(names, coord_dtype)`.

```python
>>> dtype_template = [(name, np.float) for name in cs.coord_names]
>>> dtype_should_be = np.dtype(dtype_template)
>>> cs.dtype == dtype_should_be
True
```

Two CoordinateSystems are equal if they have the same dtype and the same names and the same name.

```python
>>> another_cs = CoordinateSystem(names, 'not irrelevant', np.float)
>>> cs == another_cs
False
>>> cs.dtype == another_cs.dtype
True
>>> cs.name == another_cs.name
False
```

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>coord_dtype</code></td>
<td>Return the index of a given named coordinate.</td>
</tr>
<tr>
<td><code>index(coord_name)</code></td>
<td>Similarity is defined by self.dtype, ignoring name</td>
</tr>
<tr>
<td><code>similar_to(other)</code></td>
<td>Similarity is defined by self.dtype, ignoring name</td>
</tr>
</tbody>
</table>

#### `__init__` (coord_names, name='', coord_dtype=type('float'))

Create a coordinate system with a given name and coordinate names.

The CoordinateSystem has two `dtype` attributes:

1. self.coord_dtype is the `dtype` of the individual coordinate values
2. self.dtype is the recarray `dtype` for the CoordinateSystem which combines the `coord_names` and the `coord_dtype`. This functions as the description of the CoordinateSystem.

**Parameters**

- **coord_names**: iterable
  A sequence of coordinate names.
- **name**: string, optional
  The name of the coordinate system
- **coord_dtype**: np.dtype, optional
  The `dtype` of the `coord_names`. This should be a built-in numpy scalar `dtype`. (default is np.float). The value can by anything that can be passed to the np.dtype construc-
tor. For example np.float, np.dtype(np.float) or f8 all result in the same coord_dtype.

Examples

```python
>>> c = CoordinateSystem('ij', name='input')
>>> print c
CoordinateSystem(coord_names=('i', 'j'), name='input', coord_dtype=float64)
>>> c.coord_dtype
dtype('float64')
```

coord_dtype
alias of float64
coord_names = ('x', 'y', 'z')
dtype = dtype([('x', '<f8'), ('y', '<f8'), ('z', '<f8')])

```python
>>> c = CoordinateSystem('ij', name='input')
>>> c.index('i')
0
>>> c.index('j')
1
```

name = 'world-LPI'
ndim = 3

```python
similar_to(other)
Similarity is defined by self.dtype, ignoring name
```

Parameters other: CoordinateSystem

The object to be compared with

Returns tf: bool:

74.2.4 CoordinateSystemError

```python
class nipy.core.reference.coordinate_system.CoordinateSystemError
Bases: exceptions.Exception

__init__()
x.__init__(...) initializes x; see help(type(x)) for signature
```

args
message

74.3 Functions

```python
nipy.core.reference.coordinate_system.is_coordsys(obj)
Test if obj has the CoordinateSystem API
```

Parameters obj: object
Object to test

Returns tf: bool

True if obj has the coordinate system API

Examples

>>> is_coordsys(CoordinateSystem('xyz'))
True

>>> is_coordsys(CoordSysMaker('ikj'))
False

nipy.core.reference.coordinate_system.is_coordsys_maker(obj)
Test if obj has the CoordSysMaker API

Parameters obj: object

Object to test

Returns tf: bool

True if obj has the coordinate system API

Examples

>>> is_coordsys_maker(CoordSysMaker('ikj'))
True

>>> is_coordsys_maker(CoordinateSystem('xyz'))
False

nipy.core.reference.coordinate_system.product(*coord_systems, **kwargs)
Create the product of a sequence of CoordinateSystems.

The coord_dtype of the result will be determined by safe_dtype.

Parameters *coord_systems: sequence of CoordinateSystem

name: str

Name of output coordinate system

Returns product_coord_system: CoordinateSystem

Examples

>>> c1 = CoordinateSystem('ij', 'input', coord_dtype=np.float32)
>>> c2 = CoordinateSystem('kl', 'input', coord_dtype=np.complex)
>>> c3 = CoordinateSystem('ik', 'in3')

>>> print product(c1, c2)
CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='product', coord_dtype=complex128)

>>> print product(c1, c2, name='another name')
CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='another name', coord_dtype=complex128)
>>> product(c2, c3)
Traceback (most recent call last):
...
ValueError: coord_names must have distinct names

nipy.core.reference.coordinate_system.safe_dtype(*dtypes)
Determine a dtype to safely cast all of the given dtypes to.
Safe dtypes are valid numpy dtypes or python types which can be cast to numpy dtypes. See numpy.sctypes for a list of valid dtypes. Composite dtypes and string dtypes are not safe dtypes.

Parameters dtypes: sequence of np.dtype

Returns dtype: np.dtype

Examples

>>> c1 = CoordinateSystem('ij', 'input', coord_dtype=np.float32)
>>> c2 = CoordinateSystem('kl', 'input', coord_dtype=np.complex)
>>> safe_dtype(c1.coord_dtype, c2.coord_dtype)
dtype('complex128')

>>> # Strings are invalid dtypes
>>> safe_dtype(type('foo'))
Traceback (most recent call last):
...
TypeError: dtype must be valid numpy dtype int, uint, float, complex or object

>>> # Check for a valid dtype
>>> myarr = np.zeros(2, np.float32)
>>> myarr.dtype.isbuiltin
1

>>> # Composite dtypes are invalid
>>> mydtype = np.dtype([(‘name’, ‘S32’), (‘age’, ‘i4’)])
>>> myarr = np.zeros(2, mydtype)
>>> myarr.dtype.isbuiltin
0
>>> safe_dtype(mydtype)
Traceback (most recent call last):
...
TypeError: dtype must be valid numpy dtype int, uint, float, complex or object
75.1 Module: core.reference.slices

A set of methods to get coordinate maps which represent slices in space.

75.2 Functions

nipy.core.reference.slices.bounding_box(coordmap, shape)
Determine a valid bounding box from a CoordinateMap and a shape.

Parameters

coordmap : CoordinateMap or AffineTransform
    Containing mapping between voxel coordinates implied by shape and physical coordinates.

shape : sequence of int
    shape implying array

Returns

limits : (N,) tuple of (2,) tuples of float
    minimum and maximum coordinate values in output space (range) of coordmap. N is given by coordmap.ndim[1].

Examples

Make a 3D voxel to mni coordmap

>>> from nipy.core.api import vox2mni
>>> affine = np.array([[1, 0, 0, 2],
...                     [0, 3, 0, 4],
...                     [0, 0, 5, 6],
...                     [0, 0, 0, 1]], dtype=np.float64)
>>> A = vox2mni(affine)
>>> bounding_box(A, (30,40,20))
((2.0, 31.0), (4.0, 121.0), (6.0, 101.0))

nipy.core.reference.slices.xslice(x, y_spec, z_spec, world)
Return an LPS slice through a 3d box with x fixed.

Parameters

x : float
    The value at which x is fixed.
y_spec : sequence
   A sequence with 2 values of form ((float, float), int). The (float, float) components are
   the min and max y values; the int is the number of points.

z_spec : sequence
   As for y_spec but for z

world : str or CoordinateSystem CoordSysMaker or XYZSpace
   World 3D space to which resulting coordmap refers

Returns affine_transform : AffineTransform
   An affine transform that describes an plane in LPS coordinates with x fixed.

Examples

```python
>>> y_spec = ([-114,114], 115) # voxels of size 2 in y, starting at -114, ending at 114
>>> z_spec = ([-70,100], 86) # voxels of size 2 in z, starting at -70, ending at 100
>>> x30 = xslice(30, y_spec, z_spec, 'scanner')
>>> x30([0,0])
array([ 30., -114., -70.])
>>> x30([114,85])
array([ 30., 114., 100.])
>>> x30
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i_y', 'i_z'), name='slice', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('scanner-x=L->R', 'scanner-y=P->A', 'scanner-z=I->S'), name='scanner', coord_dtype=float64),
    affine=array([[ 0., 0., 30.],
                  [ 2., 0., -114.],
                  [ 0., 2., -70.],
                  [ 0., 0., 1.]])
)
>>> bounding_box(x30, (y_spec[1], z_spec[1]))
((30.0, 30.0), (-114.0, 114.0), (-70.0, 100.0))
```

nipy.core.reference.slices.yslice(y, x_spec, z_spec, world)
Return a slice through a 3d box with y fixed.

Parameters y : float
   The value at which y is fixed.

x_spec : sequence
   A sequence with 2 values of form ((float, float), int). The (float, float) components are
   the min and max x values; the int is the number of points.

z_spec : sequence
   As for x_spec but for z

world : str or CoordinateSystem CoordSysMaker or XYZSpace
   World 3D space to which resulting coordmap refers

Returns affine_transform : AffineTransform
   An affine transform that describes an plane in LPS coordinates with y fixed.
**Examples**

```python
>>> x_spec = ([92], 3)  # voxels of size 2 in x, starting at -92, ending at 92
>>> y_spec = ([70, 86], 3)  # voxels of size 2 in y, starting at 70, ending at 100
>>> z40 = zslice(40, x_spec, y_spec, 'unknown')
>>> z40
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i_x', 'i_y'), name='slice', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('unknown-x=L->R', 'unknown-y=P->A', 'unknown-z=I->S'), name='unknown', coord_dtype=float64),
    affine=array([[ 2., 0., -92.],
                   [ 0., 2., -114.],
                   [ 0., 0., 40.],
                   [ 0., 0., 1.]])
)
>>> z40([0,0])
array([-92., -114., 40.])
```

Nipy.core.reference.slices.zslice(z, x_spec, y_spec, world)

Return a slice through a 3D box with z fixed.

**Parameters**

- **z**: float
  
  The value at which z is fixed.

- **x_spec**: sequence
  
  A sequence with 2 values of form ((float, float), int). The (float, float) components are the min and max x values; the int is the number of points.

- **y_spec**: sequence
  
  As for x_spec but for y.

- **world**: str or CoordinateSystem CoordSysMaker or XYZSpace
  
  World 3D space to which resulting coordmap refers.

**Returns**

- **affine_transform**: AffineTransform

  An affine transform that describes a plane in LPS coordinates with z fixed.
array([ -92., -114.,  40.])
>>> z40([92,114])
array([92., 114., 40.])
>>> bounding_box(z40, (x_spec[1], y_spec[1]))
((-92.0, 92.0), (-114.0, 114.0), (40.0, 40.0))
76.1 Module: core.reference.spaces

Inheritance diagram for nipy.core.reference.spaces:

Useful neuroimaging coordinate map makers and utilities

76.2 Classes

76.2.1 AffineError

class nipy.core.reference.spaces.AffineError
    Bases: nipy.core.reference.spaces.SpaceError

    __init__(...)
    x.__init__(...) initializes x; see help(type(x)) for signature
    
    args
    message
76.2.2 AxesError

class nipy.core.reference.spaces.AxesError
    Bases: nipy.core.reference.spaces.SpaceError

    __init__(self) initializes x; see help(type(x)) for signature
        args
        message

76.2.3 SpaceError

class nipy.core.reference.spaces.SpaceError
    Bases: exceptions.Exception

    __init__(self) initializes x; see help(type(x)) for signature
        args
        message

76.2.4 SpaceTypeError

class nipy.core.reference.spaces.SpaceTypeError
    Bases: nipy.core.reference.spaces.SpaceError

    __init__(self) initializes x; see help(type(x)) for signature
        args
        message

76.2.5 XYZSpace

class nipy.core.reference.spaces.XYZSpace(name)
    Bases: object

    Class contains logic for spaces with XYZ coordinate systems
        >>> sp = XYZSpace('hijo')
        >>> print sp
        hijo: [('x', 'hijo-x=L->R'), ('y', 'hijo-y=P->A'), ('z', 'hijo-z=I->S')]
        >>> csm = sp.to_coordsys_maker()
        >>> cs = csm(3)
        >>> cs
        CoordinateSystem(coord_names=('hijo-x=L->R', 'hijo-y=P->A', 'hijo-z=I->S'), name='hijo', coord_dtypes=[int64, int64, int64])
        >>> cs in sp
        True

    Methods
Neuroimaging in Python Documentation, Release 0.3.0

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>as_map()</code></td>
<td>Return xyz names as dictionary</td>
</tr>
<tr>
<td><code>as_tuple()</code></td>
<td>Return xyz names as tuple</td>
</tr>
<tr>
<td><code>register_to(mapping)</code></td>
<td>Update mapping with key=self.x, value='x' etc pairs</td>
</tr>
<tr>
<td><code>to_coordsys_maker([extras])</code></td>
<td>Make a coordinate system maker for this space</td>
</tr>
</tbody>
</table>

### __init__(name)

**as_map()**
Return xyz names as dictionary

```python
>>> sp = XYZSpace('hijo')
>>> sorted(sp.as_map().items())
[('x', 'hijo-x=L->R'), ('y', 'hijo-y=P->A'), ('z', 'hijo-z=I->S')]
```

**as_tuple()**
Return xyz names as tuple

```python
>>> sp = XYZSpace('hijo')
>>> sp.as_tuple()
('hijo-x=L->R', 'hijo-y=P->A', 'hijo-z=I->S')
```

**register_to(mapping)**
Update mapping with key=self.x, value='x' etc pairs

The mapping will then have keys that are names we (self) identify as being x, or y, or z, values are 'x' or 'y' or 'z'.

Note that this is the opposite way round for keys, values, compared to the as_map method.

**Parameters**
- **mapping**: mapping such as a dict

**Returns**
- **None**

### Examples

```python
>>> sp = XYZSpace('hijo')
>>> mapping = {}
>>> sp.register_to(mapping)
>>> sorted(mapping.items())
[('hijo-x=L->R', 'x'), ('hijo-y=P->A', 'y'), ('hijo-z=I->S', 'z')]
```

**to_coordsys_maker([extras])**
Make a coordinate system maker for this space

**Parameters**
- **extra**: sequence names for any further axes after x, y, z

**Returns**
- **csm**: CoordinateSystemMaker

### Examples
>>> sp = XYZSpace('hijo')
>>> csm = sp.to_coordsys_maker()
>>> csm(3)
CoordinateSystem(coord_names=('hijo-x=L->R', 'hijo-y=P->A', 'hijo-z=I->S'), name='hijo', coord_dtype=float64)

x
  x-space coordinate name

x_suffix = 'x=L->R'

y
  y-space coordinate name

y_suffix = 'y=P->A'

z
  z-space coordinate name

z_suffix = 'z=I->S'

## 76.3 Functions

nipy.core.reference.spaces.get_world_cs(world_id, ndim=3, extras=’tuvw’, spaces=None)

Get world coordinate system from world_id

**Parameters**

- **world_id**: str, XYZSpace, CoordSysMaker or CoordinateSystem
  
  Object defining a world output system. If str, then should be a name of an XYZSpace in the list `spaces`.

- **ndim**: int, optional
  
  Number of dimensions in this world. Default is 3

- **extras**: sequence, optional
  
  Coordinate (axis) names for axes > 3 that are not named by `world_id`

- **spaces**: None or sequence, optional
  
  List of known (named) spaces to compare a str `world_id` to. If None, use the module level known_spaces

**Returns**

- **world_cs**: CoordinateSystem
  
  A world coordinate system

**Examples**

```python
>>> get_world_cs('mni')
CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S'), name='mni', coord_dtype=float64)

>>> get_world_cs(mni_space, 4)
CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S', 't'), name='mni', coord_dtype=float64)

>>> from nipy.core.api import CoordinateSystem
>>> get_world_cs(CoordinateSystem('xyz'))
CoordinateSystem(coord_names=('x', 'y', 'z'), name='', coord_dtype=float64)
```
**is_xyz_affable** *(coordmap, name2xyz=None)*

Return True if the coordmap has an xyz affine

- **Parameters**
  - *coordmap*: CoordinateMap instance
    Coordinate map to test
  - *name2xyz*: None or mapping, optional
    Object such that `name2xyz[ax_name]` returns ‘x’, or ‘y’ or ‘z’ or raises a KeyError for a str `ax_name`. None means use module default.

- **Returns**
  - *tf*: bool
    True if `coordmap` has an xyz affine, False otherwise

**Examples**

```python
>>> cmap = vox2mni(np.diag([2,3,4,5,1]))
>>> cmap
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='voxels', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S', 't'), name='mni', coord_dtype=float64),
    affine=array([[ 2., 0., 0., 0., 0.],
                  [ 0., 3., 0., 0., 0.],
                  [ 0., 0., 4., 0., 0.],
                  [ 0., 0., 0., 5., 0.],
                  [ 0., 0., 0., 0., 1.]]))

>>> is_xyz_affable(cmap)
True
```

```python
>>> time0_cmap = cmap.reordered_domain([3,0,1,2])
>>> time0_cmap
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('l', 'i', 'j', 'k'), name='voxels', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S', 't'), name='mni', coord_dtype=float64),
    affine=array([[ 0., 2., 0., 0., 0.],
                  [ 0., 0., 3., 0., 0.],
                  [ 0., 0., 0., 4., 0.],
                  [ 5., 0., 0., 0., 0.],
                  [ 0., 0., 0., 0., 1.]]))

>>> is_xyz_affable(time0_cmap)
False
```

**is_xyz_space** *(obj)*

True if `obj` appears to be an XYZ space definition

**known_space** *(obj, spaces=None)*

If `obj` is in a known space, return the space, otherwise return None

- **Parameters**
  - *obj*: object
    Object that can be tested against an XYZSpace with `obj in sp`
  - *spaces*: None or sequence, optional
    spaces to test against. If None, use the module level `known_spaces` list to test against.

- **Returns**
  - *sp*: None or XYZSpace
If `obj` is not in any of the `known_spaces`, return None. Otherwise return the first matching space in `known_spaces`.

### Examples

```python
>>> from nipy.core.api import CoordinateSystem
>>> sp0 = XYZSpace('hijo')
>>> sp1 = XYZSpace('hija')

Make a matching coordinate system
```nn
```python
>>> cs = sp0.to_coordsys_maker()(3)
Test whether this coordinate system is in either of (sp0, sp1)
```nn
```python
>>> known_space(cs, (sp0, sp1))
XYZSpace('hijo')

So, yes, it’s in sp0. How about another generic CoordinateSystem?
```nn
```python
>>> known_space(CoordinateSystem('xyz'), (sp0, sp1)) is None
True

So, no, that is not in either of (sp0, sp1)
```

### `nipy.core.reference.spaces.xyz_affine(coordmap, name2xyz=None)`
Return (4, 4) affine mapping voxel coordinates to XYZ from `coordmap`

If no (4, 4) affine “makes sense”(TM) for this `coordmap` then raise errors listed below. A (4, 4) affine makes sense if the first three output axes are recognizably X, Y, and Z in that order AND they there are corresponding input dimensions, AND the corresponding input dimensions are the first three input dimension (in any order). Thus the input axes have to be 3D.

- **Parameters**
  - `coordmap`: CoordinateMap instance
  - `name2xyz`: None or mapping, optional

    Object such that `name2xyz[ax_name]` returns ‘x’, or ‘y’ or ‘z’ or raises a KeyError for a str `ax_name`. None means use module default.

- **Returns**
  - `xyz_aff`: (4,4) array

    voxel to X, Y, Z affine mapping

- **Raises**
  - `SpaceTypeError`: if this is not an affine coordinate map
  - `AxesError`: if not all of x, y, z recognized in `coordmap` output, or they are in the wrong order, or the x, y, z axes do not correspond to the first:

    three input axes.

  - `AffineError`: if axes dropped from the affine contribute to x, y, z coordinates.

### Notes

We could also try and “make sense” (TM) of a coordmap that had X, Y and Z outputs, but not in that order, nor all in the first three axes. In that case we could just permute the affine to get the output order we need. But, that
could become confusing if the returned affine has different output coordinates than the passed coordmap. And it’s more complicated. So, let’s not do that for now.

Examples

```
>>> cmap = vox2mni(np.diag([2,3,4,5,1]))
>>> cmap
AffineTransform(
    function_domain=CoordinateSystem(coord_names=('i', 'j', 'k', 'l'), name='voxels', coord_dtype=float64),
    function_range=CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S', 't'), name='mni', coord_dtype=float64),
    affine=array([[ 2., 0., 0., 0., 0.],
                  [ 0., 3., 0., 0., 0.],
                  [ 0., 0., 4., 0., 0.],
                  [ 0., 0., 5., 0.],
                  [ 0., 0., 0., 1.]]))
```

```
nipy.core.reference.spaces.xyz_order(coordsys, name2xyz=None)
Vector of orders for sorting coordsys axes in xyz first order

Parameters
coordsys : CoordinateSystem instance
name2xyz : None or mapping, optional
    Object such that name2xyz[ax_name] returns ‘x’, or ‘y’ or ‘z’ or raises a KeyError for a str ax_name. None means use module default.

Returns
xyz_order : list
    Ordering of axes to get xyz first ordering. See the examples.

Raises
AxesError : if there are not all of x, y and z axes
```

Examples

```
>>> from nipy.core.api import CoordinateSystem
>>> xyzt_cs = mni_csm(4) # cooridsys with t (time) last
>>> xyzt_cs
CoordinateSystem(coord_names=('mni-x=L->R', 'mni-y=P->A', 'mni-z=I->S', 't'), name='mni', coord_dtype=float64)
>>> xyz_order(xyzt_cs)
[0, 1, 2, 3]
>>> tzyx_cs = CoordinateSystem(xyzt_cs.coord_names[::-1], 'reversed')
>>> tzyx_cs
CoordinateSystem(coord_names=('t', 'mni-z=I->S', 'mni-y=P->A', 'mni-x=L->R'), name='reversed', coord_dtype=float64)
>>> xyz_order(tzyx_cs)
[3, 2, 1, 0]
```
77.1 Module: core.utils.generators

This module defines a few common generators for slicing over arrays. They are defined on ndarray, so they do not depend on Image.

- data_generator: return (item, data[item]) tuples from an iterable object
- slice_generator: return slices through an ndarray, possibly over many indices
- f_generator: return a generator that applies a function to the output of another generator

The above three generators return 2-tuples.

- write_data: write the output of a generator to an ndarray
- parcels: return binary array of the unique components of data

77.2 Functions

nipy.core.utils.generators.data_generator(data, iterable=None)
Return generator for [(i, data[i]) for i in iterable]
If iterable is None, it defaults to range(data.shape[0])

Examples

```python
>>> a = np.asarray([[True, False], [False, True]])
>>> b = np.asarray([[False, False], [True, False]])

>>> for i, d in data_generator(np.asarray([[1,2],[3,4]]), [a,b]):
...     print d
...     [1 4]
[3]
```

nipy.core.utils.generators.f_generator(f, iterable)
Return a generator for [(i, f(x)) for i, x in iterable]
Examples

```python
>>> for i, d in f_generator(lambda x: x**2, data_generator([[1,2],[3,4]])):
...     print i, d
...
0 [1 4]
1 [ 9 16]
```

```
nipy.core.utils.generators.matrix_generator(img)
From a generator of items (i, r), return (i, rp) where rp is a 2d array with rp.shape = (r.shape[0], prod(r.shape[1:]))
```

```
nipy.core.utils.generators.parcels(data, labels=None, exclude=[]) Return a generator for [data == label for label in labels]
If labels is None, labels = numpy.unique(data). Each label in labels can be a sequence, in which case the value returned for that label union:
[numpy.equal(data, l) for l in label]
```

Examples

```python
>>> for p in parcels([[1,1],[2,1]]):
...     print p
...
[[ True  True]
 [False  True]]
[[False False]
 [ True False]]
>>> for p in parcels([[1,1],[2,3]], labels=[2,3]):
...     print p
...
[[False False]
 [ True False]]
[[False False]
 [False True]]
>>> for p in parcels([[1,1],[2,3]], labels=|(2,3),2|):
...     print p
...
[[False False]
 [ True True]]
[[False False]
 [True True]]
```

```
nipy.core.utils.generators.shape_generator(img, shape)
From a generator of items (i, r), return (i, r.reshape(shape))
```

```
nipy.core.utils.generators.slice_generator(data, axis=0)
Return generator for yielding slices along axis
```

Examples

```python
>>> for i,d in slice_generator([[1,2],[3,4]]):
...     print i, d
...
(0,) [1 2]
(1,) [3 4]
```
>>> for i, d in slice_generator([[1,2],[3,4]], axis=1):
    ...     print i, d
...
(slice(None, None, None), 0) [1 3]
(slice(None, None, None), 1) [2 4]

*nipy.core.utils.generators.slice_parcel*(data, labels=None, axis=0)

A generator for slicing through parcels and slices of data...

hmmm... a better description is needed

>>> x=np.array([[0,0,0,1],[0,1,0,1],[2,2,0,1]])
>>> for a in slice_parcel(x):
    ...     print a, x[a]
...
((0,), array([ True, True, True, False], dtype=bool)) [0 0 0]
((0,), array([False, False, False, True], dtype=bool)) [1]
((1,), array([ True, False, True, False], dtype=bool)) [0 0]
((1,), array([False, False, True, False], dtype=bool)) [1 1]
((2,), array([False, False, True, False], dtype=bool)) [0]
((2,), array([False, False, False, True], dtype=bool)) [1]
((2,), array([ True, True, False, False], dtype=bool)) [2 2]

>>> for a in slice_parcel(x, axis=1):
    ...     b, c = a
    ...     print a, x[b][c]
...
((slice(None, None, None), 0), array([ True, True, False], dtype=bool)) [0 0]
((slice(None, None, None), 0), array([False, False, True], dtype=bool)) [2]
((slice(None, None, None), 1), array([ True, False, False], dtype=bool)) [0]
((slice(None, None, None), 1), array([False, True, False], dtype=bool)) [1]
((slice(None, None, None), 1), array([False, False, True], dtype=bool)) [2]
((slice(None, None, None), 2), array([ True, True, True], dtype=bool)) [0 0 0]
((slice(None, None, None), 3), array([ True, True, True], dtype=bool)) [1 1 1]

*nipy.core.utils.generators.write_data*(output, iterable)

Write (index, data) iterable to output

Write some data to output. Iterable should return 2-tuples of the form index, data such that:

output[index] = data

makes sense.

Examples

>>> a=np.zeros((2,2))
>>> write_data(a, data_generator(np.asarray([[1,2],[3,4]])))
>>> a
array([[ 1.,  2.],
       [ 3.,  4.]])
# Interfaces: matlab

## 78.1 Module: interfaces.matlab

General matlab interface code

This is for nipy convenience. If you’re doing heavy matlab interfacing, please use NiPype instead:

http://nipy.org/nipype

## 78.2 Functions

[nipy.interfaces.matlab.mlab_tempfile(dir=None)](http://nipy.org/nipype)

Returns a temporary file-like object with valid matlab name.

The file name is accessible as the .name attribute of the returned object. The caller is responsible for closing the returned object, at which time the underlying file gets deleted from the filesystem.

**Parameters**

- `dir`: str
  
  A path to use as the starting directory. Note that this directory must already exist, it is NOT created if it doesn’t (in that case, OSErrror is raised instead).

**Returns**

- `f`: file-like object

**Examples**

```python
>>> f = mlab_tempfile()
>>> pth, fname = os.path.split(f.name)
>>> '-' not in fname
True
>>> f.close()
```

[nipy.interfaces.matlab.run_matlab(cmd)](http://nipy.org/nipype)

[nipy.interfaces.matlab.run_matlab_script(script_lines, script_name='pyscript')](http://nipy.org/nipype)

Put multiline matlab script into script file and run
79.1 Module: `interfaces.spm`

Inheritance diagram for `nipy.interfaces.spm`:

```
interfaces.spm.SpmInfo
```

Interfaces to SPM

79.2 Class

79.3 SpmInfo

```python
class nipy.interfaces.spm.SpmInfo
    Bases: object

    Methods

    spm_path()

    __init__()
        x.__init__(...) initializes x; see help(type(x)) for signature

    static spm_path()
```

```
79.4 Functions

nipy.interfaces.spm.fltcols(vals)
    Trivial little function to make 1xN float vector

nipy.interfaces.spm.fname_presuffix(fname, prefix='', suffix='', use_ext=True)

nipy.interfaces.spm.fnames_presuffix(fnames, prefix='', suffix=')

nipy.interfaces.spm.make_job(jobtype, jobname, contents)

nipy.interfaces.spm.run_jobdef(jobdef)

nipy.interfaces.spm.scans_for_fname(fname)

nipy.interfaces.spm.scans_for_fnames(fnames)
80.1 Module: io.files

The io.files module provides basic functions for working with file-based images in nipy.

- load : load an image from a file
- save : save an image to a file

80.1.1 Examples

See documentation for load and save functions for worked examples.

80.2 Functions

nipy.io.files.as_image(image_input)

Load image from filename or pass through image instance

Parameters

image_input : str or Image instance

image or string filename of image. If a string, load image and return. If an image, pass through without modification

Returns

img : Image or Image-like instance

Input object if image_input seemed to be an image, loaded Image object if image_input was a string.

Raises

TypeError : if neither string nor image-like passed

Examples

```python
>>> from nipy.testing import anatfile
>>> from nipy.io.api import load_image
>>> img = as_image(anatfile)
>>> img2 = as_image(img)
>>> img2 is img
True
```

nipy.io.files.load(filename)

Load an image from the given filename.
Parameters filename : string

Should resolve to a complete filename path.

Returns image : An Image object

If successful, a new Image object is returned.

See Also:

save_image function for saving images

Image image object

Examples

```python
>>> from nipy.io.api import load_image
>>> from nipy.testing import anatfile
>>> img = load_image(anatfile)
>>> img.shape
(33, 41, 25)
```

```
nipy.io.files.save(img, filename, dtype_from='data')
```

Write the image to a file.

Parameters img : An Image object

filename : string

Should be a valid filename.

dtype_from : {'data', 'header'} or dtype specifier, optional

Method of setting dtype to save data to disk. Value of ‘data’ (default), means use data
dtype to save. ‘header’ means use data dtype specified in header, if available, other-
wise use data dtype. Can also be any valid specifier for a numpy dtype, e.g. ‘i4’,
np.float32. Not every format supports every dtype, so some values of this param-
eter or data dtypes will raise errors.

Returns image : An Image object

Possibly modified by saving.

See Also:

load_image function for loading images

Image image object

Notes

Filetype is determined by the file extension in ‘filename’. Currently the following filetypes are supported:

- Nifti single file : ['.nii', '.nii.gz']
- Nifti file pair : ['.hdr', '.hdr.gz']
- SPM Analyze : ['.img', '.img.gz']
Examples

Make a temporary directory to store files

```python
>>> import os
>>> from tempfile import mkdtemp
>>> tmpdir = mkdtemp()
```

Make some some files and save them

```python
>>> import numpy as np
>>> from nipy.core.api import Image, AffineTransform
>>> from nipy.io.api import save_image

>>> data = np.zeros((91,109,91), dtype=np.uint8)
>>> cmap = AffineTransform('kji', 'zxy', np.eye(4))
>>> img = Image(data, cmap)
>>> fname1 = os.path.join(tmpdir, 'img1.nii.gz')
>>> saved_img1 = save_image(img, fname1)

>>> saved_img1.shape
(91, 109, 91)
>>> fname2 = os.path.join(tmpdir, 'img2.img.gz')
>>> saved_img2 = save_image(img, fname2)

>>> saved_img2.shape
(91, 109, 91)
>>> fname = 'test.mnc'
>>> saved_image3 = save_image(img, fname)
```

```
Traceback (most recent call last):
...
ValueError: Sorry, we cannot yet save as format "minc"
```

Finally, we clear up our temporary files:

```python
>>> import shutil
>>> shutil.rmtree(tmpdir)
```
81.1 Module: io.nifti_ref

Inheritance diagram for nipy.io.nifti_ref:

io.nifti_ref.NiftiError

An implementation of some of the NIFTI conventions as described in:
A version of the same file is in the nibabel repository at doc/source/external/nifti1.h.

81.1.1 Background

We (nipystas) make an explicit distinction between:

- an input coordinate system of an image (the array == voxel coordinates)
- output coordinate system (usually millimeters in some world for space, seconds for time)
- the mapping between the two.

The collection of these three is the coordmap attribute of a NIPY image.

There is no constraint that the number of input and output coordinates should be the same.

We don’t specify the units of our output coordinate system, but assume spatial units are millimeters and time units are seconds.

NIFTI is mostly less explicit, but more constrained.

NIFTI input coordinate system

NIFTI files can have up to seven voxel dimensions (7 axes in the input coordinate system).
The first 3 voxel dimensions of a NIFTI file must be spatial but can be in any order in relationship to directions in mm space (the output coordinate system).

The 4th voxel dimension is assumed to be time. In particular, if you have some other meaning for a non-spatial dimension, the NIFTI standard suggests you set the length of the 4th dimension to be 1, and use the 5th dimension of the image instead, and set the NIFTI “intent” fields to state the meaning. If the intent field is set correctly then it should be possible to set meaningful input coordinate axis names for dimensions > (0, 1, 2).

There’s a wrinkle to the 4th axis is time story; the xyzt_units field in the NIFTI header can specify the 4th dimension units as Hz (frequency), PPM (concentration) or Radians / second.

NIFTI also has a ‘dim_info’ header attribute that optionally specifies that 0 or more of the first three voxel axes are ‘frequency’, ‘phase’ or ‘slice’. These terms refer to 2D MRI acquisition encoding, where ‘slice’s are collected sequentially, and the two remaining dimensions arose from frequency and phase encoding. The dim_info fields are often not set. 3D acquisitions don’t have a ‘slice’ dimension.

**NIFTI output coordinate system**

In the NIFTI specification, the order of the output coordinates (at least the first 3) are fixed to be what might be called RAS+, that is (‘x=L->R’, ‘y=P->A’, ‘z=I->S’). This RAS+ output order is not allowed to change and there is no way of specifying such a change in the NIFTI header.

The world in which these RAS+ X, Y, Z axes exist can be one of the recognized spaces, which are: scanner, aligned (to another file’s world space), Talairach, MNI 152 (aligned to the MNI 152 atlas).

By implication, the 4th output dimension is likely to be seconds (given the 4th input dimension is likely time), but there’s a field xyzt_units (see above) that can be used to imply the 4th output dimension is actually frequency, concentration or angular velocity.

**NIFTI input / output mapping**

NIFTI stores the relationship between the first 3 (spatial) voxel axes and the RAS+ coordinates in an XYZ affine. This is a homogenous coordinate affine, hence 4 by 4 for 3 (spatial) dimensions.

NIFTI also stores “pixel dimensions” in a pixdim field. This can give you scaling for individual axes. We ignore the values of pixdim for the first 3 axes if we have a full (“sform”) affine stored in the header, otherwise they form part of the affine above. pixdim’[3:] provide voxel to output scalings for later axes. The units for the 4th dimension can come from ‘xyzt_units as above.

We take the convention that the output coordinate names are (‘x=L->R’, ‘y=P->A’, ‘z=I->S’, ‘t’, ‘u’, ‘v’, ‘w’) unless there is no time axis (see below) in which case we just omit ‘t’. The first 3 axes are also named after the output space (‘scanner-x=L->R’, ‘mni-x=L-R’ etc).

The input axes are ‘ijktuvw’ unless there is no time axis (see below), in which case they are ‘ijklvw’ (remember, NIFTI only allows 7 dimensions, and one is used up by the time length 1 axis).

**Time-like axes**

A time-like axis is an axis that is any of time, Hz, PPM or radians / second.

We recognize time in a NIPY coordinate map by an input or an output axis named ‘t’ or ‘time’. If it’s an output axis we work out the corresponding input axis.

A Hz axis can be called ‘hz’ or ‘frequency-hz’.

A PPM axis can be called ‘ppm’ or ‘concentration-ppm’.

A radians / second axis can be called ‘rads’ or ‘radians/s’.
Does this NIFTI image have a time-like axis?

We take there to be no time axis if there are only three NIFTI dimensions, or if:

- the length of the fourth NIFTI dimension is 1 AND
- There are more than four dimensions AND
- The `xyzt_units` field does not indicate time or time-like units.

### 81.1.2 What we do about all this

For saving a NIPY image to NIFTI, see the docstring for `nipy2nifti()`. For loading a NIFTI image to NIPY, see the docstring for `nifti2nipy()`.

### 81.2 Class

#### 81.3 NiftiError

class `nipy.io.nifti_ref.NiftiError`

    Bases: exceptions.Exception

    __init__(self, *args, **kwargs)

        x.__init__(...) initializes x; see help(type(x)) for signature

        args

        message

#### 81.4 Functions

```
import nipy.io.nifti_ref

nifti2nipy(ni_img)
```

Return NIPY image from NIFTI image `ni_img`

**Parameters**

- `ni_img` : nibabel.Nifti1Image

**Returns**

- `img` : Image

   nipy image

**Raises**

- NiftiError : if image is < 3D

**Notes**

Lacking any other information, we take the input coordinate names for axes 0:7 to be (`i`, `j`, `k`, `t`, `u`, `v`, `w`).

If the image is 1D or 2D then we have a problem. If there’s a defined (sform, qform) affine, this has 3 input dimensions, and we have to guess what the extra input dimensions are. If we don’t have a defined affine, we don’t know what the output dimensions are. For example, if the image is 2D, and we don’t have an affine, are these X and Y or X and Z or Y and Z? In the presence of ambiguity, resist the temptation to guess - raise a NiftiError.
If there is a time-like axis, name the input and corresponding output axis for the type of axis (‘t’, ‘hz’, ‘ppm’, ‘rads’).

Otherwise remove the ‘t’ axis from both input and output names, and squeeze the length 1 dimension from the input data.

If there’s a ‘t’ axis get toffset and put into affine at position [3, -1].

If dim_info is set coherently, set input axis names to ‘slice’, ‘freq’, ‘phase’ from dim_info.


We construct the N-D affine by taking the XYZ affine and adding scaling diagonal elements from pixdim.

If the space units in NIFTI xyzt_units are ‘microns’ or ‘meters’ we adjust the affine to mm units, but warn because this might be a mistake.

If the time units in NIFTI xyzt_units are ‘msec’ or ‘usec’, scale the time axis pixdim values accordingly.

Ignore the intent-related fields for now, but warn that we are doing so if there appears to be specific information in there.

nipy.io.nifti_ref.nipy2nifti (img, data_dtype=None, strict=None, fix0=True)

Return NIFTI image from nipy image img

Parameters  

**img** : object

An object, usually a NIPY Image, having attributes coordmap and shape

**data_dtype** : None or dtype specifier

None means try and use header dtype, otherwise try and use data dtype, otherwise use np.float32. A dtype specifier means set the header output data dtype using np.dtype(data_dtype).

**strict** : bool, optional

Whether to use strict checking of input image for creating NIFTI

**fix0** : bool, optional:

Whether to fix potential 0 column / row in affine. This option only used when trying to find time etc axes in the coordmap output names. In order to find matching input names, we need to use the corresponding rows and columns in the affine. Sometimes time, in particular, has 0 scaling, and thus all 0 in the corresponding row / column. In that case it’s hard to work out which input corresponds. If fix0 is True, and there is only one all zero (matrix part of the) affine row, and only one all zero (matrix part of the) affine column, fix scaling for that combination to zero, assuming this a zero scaling for time.

Returns **ni_img** : nibabel.Nifti1Image

NIFTI image

Raises  

NiftiError: if space axes not orthogonal to non-space axes : 

NiftiError: if non-space axes not orthogonal to each other :

NiftiError: if ‘img’ output space does not match named spaces in NIFTI :

NiftiError: if input image has more than 7 dimensions :

NiftiError: if input image has 7 dimensions, but no time dimension, because :

we need to add an extra 1 length axis at position 3
**NiftiError**: if we find a time-like input axis but the matching output axis:

is a different time-like.

**NiftiError**: if we find a time-like output axis but the matching input axis:

is a different time-like.

**NiftiError**: if we find a time output axis and there are non-zero non-spatial:

offsets in the affine, but we can’t find a corresponding input axis.

**Notes**

First, we need to create a valid XYZ Affine. We check if this can be done by checking if there are recognizable X, Y, Z output axes and corresponding input (voxel) axes. This requires the input image to be at least 3D. If we find these requirements, we reorder the image axes to have XYZ output axes and 3 spatial input axes first, and get the corresponding XYZ affine.

If the spatial dimensions are not orthogonal to the non-spatial dimensions, raise a NiftiError.

If the non-spatial dimensions are not orthogonal to each other, raise a NiftiError.

We check if the XYZ output fits with the NIFTI named spaces of scanner, aligned, Talairach, MNI. If so, set the NIFTI code and qform, sform accordingly. If the space corresponds to ‘unknown’ then we must set the NIFTI transform codes to 0, and the affine must match the affine we will get from loading the NIFTI with no qform, sform. If not, we’re going to lose information in the affine, and raise an error.

If any of the first three input axes are named ('slice’, ‘freq’, ‘phase’) set the `dim_info` field accordingly.

Set the `xyzt_units` field to indicate millimeters and seconds, if there is a ‘t’ axis, otherwise millimeters and 0 (unknown).

We look to see if we have a time-like axis in the inputs or the outputs. A time-like axis has labels ‘t’, ‘hz’, ‘ppm’, ‘rads’. If we have an axis ‘t’ in the inputs and the outputs, check they either correspond, or both inputs and output correspond with no other axis, otherwise raise NiftiError. Do the same check for ‘hz’, then ‘ppm’, then ‘rads’.

If we do have a time-like axis, roll that axis to be the 4th axis. If this axis is actually time, take the `affine[3, -1]` and put into the `toffset` field. If there’s no time-like axis, but there are other non-spatial axes, make a length 1 4th array axis to indicate this.

If the resulting NIFTI image has more than 7 dimensions, raise a NiftiError.

Set pixdim for axes >= 3 using vector length of corresponding affine columns.

We don’t set the intent-related fields for now.
82.1 Module: labs.bindings.benchmarks.bench_numpy

82.2 Functions

nipy.labs.bindings.benchmarks.bench_numpy.bench_copy_vector_contiguous()
nipy.labs.bindings.benchmarks.bench_numpy.bench_copy_vector_strided()
nipy.labs.bindings.benchmarks.bench_numpy.time_copy_vector(x)
nipy.labs.bindings.benchmarks.bench_numpy.time_ratio(t0, t1)
CHAPTER
EIGHTYTHREE

LABS.DATASETS.CONVERTERS

83.1 Module: labs.datasets.converters

Conversion mechanisms for IO and interaction between volumetric datasets and other type of neuroimaging data.

83.2 Functions

nipy.labs.datasets.converters.as_volume_img(obj, copy=True, squeeze=True, world_space=None)

Convert the input to a VolumeImg.

Parameters

- **obj**: filename, pynifti or brifti object, or volume dataset.
  - Input object, in any form that can be converted to a VolumeImg. This includes Nifti filenames, pynifti or brifti objects, or other volumetric dataset objects.

- **copy**: boolean, optional
  - If copy is True, the data and affine arrays are copied, elsewhere a view is taken.

- **squeeze**: boolean, optional
  - If squeeze is True, the data array is squeeze on for dimensions above 3.

- **world_space**: string or None, optional
  - An optional specification of the world space, to override that given by the image.

Returns

- **volume_img**: VolumeImg object
  - A VolumeImg object containing the data. The metadata is kept as much as possible in the metadata attribute.

Notes

The world space might not be correctly defined by the input object (in particular, when loading data from disk). In this case, you can correct it manually using the world_space keyword argument.

For pynifti objects, the data is transposed.

nipy.labs.datasets.converters.save(filename, obj)

Save an nipy image object to a file.
84.1 Module: labs.datasets.transforms.affine_transform

Inheritance diagram for nipy.labs.datasets.transforms.affine_transform:

```
transforms.transform.Transform -> transforms.affine_transform.AffineTransform
```

The AffineTransform class

84.2 AffineTransform

```python
class nipy.labs.datasets.transforms.affine_transform.AffineTransform(input_space, output_space, affine):
```

Bases: nipy.labs.datasets.transforms.transform.Transform

A transformation from an input 3D space to an output 3D space defined by an affine matrix.

It is defined by the affine matrix , and the name of the input and output spaces.

Attributes

```
 affine
```

Methods

```
composed_with(transform) Returns a new transform obtained by composing this transform with the one provided.
get_inverse() Return the inverse transform.
inverse_mapping(x, y, z) Transform the given coordinate from output space to input space.
```
**__init__(input_space, output_space, affine)__**

Create a new affine transform object.

- **Parameters**
  - **input_space**: string
    - Name of the input space
  - **output_space**: string
    - Name of the output space
  - **affine**: 4x4 ndarray
    - Affine matrix giving the coordinate mapping between the input and output space.

- **affine** = None

**composed_with(transform)**

Returns a new transform obtained by composing this transform with the one provided.

- **Parameters**
  - **transform**: nipy.core.transforms.transform object
    - The transform to compose with.

**get_inverse()**

Return the inverse transform.

- **input_space** = ‘’

**inverse_mapping(x, y, z)**

Transform the given coordinate from output space to input space.

- **Parameters**
  - **x**: number or ndarray
    - The x coordinates
  - **y**: number or ndarray
    - The y coordinates
  - **z**: number or ndarray
    - The z coordinates

**mapping(x, y, z)**

Transform the given coordinate from input space to output space.

- **Parameters**
  - **x**: number or ndarray
    - The x coordinates
  - **y**: number or ndarray
    - The y coordinates
  - **z**: number or ndarray
    - The z coordinates

- **output_space** = ‘’
85.1 Module: `labs.datasets.transforms.affine_utils`

Functions working with affine transformation matrices.

85.2 Functions

`nipy.labs.datasets.transforms.affine_utils.apply_affine(x, y, z, affine)`

Apply the affine matrix to the given coordinate.

Parameters:
- `x`: number or ndarray
  - The x coordinates
- `y`: number or ndarray
  - The y coordinates
- `z`: number or ndarray
  - The z coordinates
- `affine`: 4x4 ndarray
  - The affine matrix of the transformation

`nipy.labs.datasets.transforms.affine_utils.from_matrix_vector(matrix, vector)`

Combine a rotation matrix and translation vector into a transform in homogeneous coordinates.

Parameters:
- `matrix`: ndarray
  - An NxN array representing the rotation matrix.
- `vector`: ndarray
  - A 1xN array representing the translation.

Returns:
- `xform`: ndarray
  - An N+1xN+1 transform matrix.

See Also:
- `to_matrix_vector`
nipy.labs.datasets.transforms.affine_utils.get_bounds(shape, affine)
    Return the world-space bounds occupied by an array given an affine.

nipy.labs.datasets.transforms.affine_utils.to_matrix_vector(transform)
    Split a transform into it’s matrix and vector components.
    The transformation must be represented in homogeneous coordinates and is split into it’s rotation matrix and translation vector components.

    Parameters
    
        transform : ndarray
            Transform matrix in homogeneous coordinates. Example, a 4x4 transform representing rotations and translations in 3 dimensions.

    Returns
    
        matrix, vector : ndarray
            The matrix and vector components of the transform matrix. For an NxN transform, matrix will be N-1xN-1 and vector will be 1xN-1.

See Also:

from_matrix_vector
LABS.DATASETS.TRANSFORMS.TRANSFORM

86.1 Module: labs.datasets.transforms.transform

Inheritance diagram for `nipy.labs.datasets.transforms.transform`:

```
transforms.transform.CompositionError

transforms.transform.Transform
```

The base Transform class.
This class defines the Transform interface and can be subclassed to define more clever composition logic.

86.2 Classes

86.2.1 CompositionError

```python
class nipy.labs.datasets.transforms.transform.CompositionError
    Bases: exceptions.Exception
    The Exception raised when composing transforms with non matching respective input
    and output word spaces.
    __init__(self, message)
        x.__init__(...) initializes x; see help(type(x)) for signature
        message
```

86.2.2 Transform

class nipy.labs.datasets.transforms.transform.Transform(input_space, output_space, mapping=None, inverse_mapping=None):

    Bases: object

    A transform is a representation of a transformation from one 3D space to another. It is composed of a coordinate mapping, or its inverse, as well as the name of the input and output spaces.

    The Transform class is the base class for transformations and defines the transform object API.

Attributes

    inverse_mapping
    mapping

Methods

    composed_with(transform) Returns a new transform obtained by composing this transform with the one provided.

    get_inverse() Return the inverse transform.

    __init__(input_space, output_space, mapping=None, inverse_mapping=None)
    Create a new transform object.

    Parameters

        mapping: callable f(x, y, z):
            Callable mapping coordinates from the input space to the output space. It should take 3 numbers or arrays, and return 3 numbers or arrays of the same shape.

        inverse_mapping: callable f(x, y, z):
            Callable mapping coordinates from the output space to the input space. It should take 3 numbers or arrays, and return 3 numbers or arrays of the same shape.

        input_space: string:
            Name of the input space

        output_space: string:
            Name of the output space

Notes

    You need to supply either the mapping or the inverse mapping.

    composed_with(transform)
    Returns a new transform obtained by composing this transform with the one provided.

    Parameters

        transform: nipy.core.transforms.transform object:
            The transform to compose with.

    get_inverse() Return the inverse transform.
```python
input_space = ''
inverse_mapping = None
mapping = None
output_space = ''
```
LABS.DATASETS.VOLUMES.VOLUME_DATA

87.1 Module: labs.datasets.volumes.volume_data

Inheritance diagram for nipy.labs.datasets.volumes.volume_data:

volumes.volume_field.VolumeField  volumes.volume_data.VolumeData

The volume data class

This class represents indexable data embedded in a 3D space

87.2 VolumeData

class nipy.labs.datasets.volumes.volume_data.VolumeData
Bases: nipy.labs.datasets.volumes.volume_field.VolumeField

A class representing data embedded in a 3D space

This object has data stored in an array like, that knows how it is mapped to a 3D “real-world space”, and how it can change real-world coordinate system.

Notes

The data is stored in an undefined way: prescalings might need to be applied to it before using it, or the data might be loaded on demand. The best practice to access the data is not to access the _data attribute, but to use the get_data method.
Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>world_space</td>
<td>World space the data is embedded in. For instance mni152.</td>
</tr>
<tr>
<td>metadata</td>
<td>Optional, user-defined, dictionary used to carry around extra information about the data as it goes through transformations. The class consistency of this information is not maintained as the data is modified.</td>
</tr>
<tr>
<td>_data</td>
<td>Private pointer to the data.</td>
</tr>
</tbody>
</table>

Methods

- `as_volume_img([affine, shape, ...])`: Resample the image to be an image with the data points lying on a regular grid with an affine mapping to the word space (a nipy VolumeImg).
  
  **Parameters**
  
  - **affine**: 4x4 or 3x3 ndarray, optional
    
    Affine of the new voxel grid or transform object pointing to the new voxel coordinate grid. If a 3x3 ndarray is given, it is considered to be the rotation part of the affine, and the best possible bounding box is calculated, in this case, the shape argument is not used. If None is given, a default affine is provided by the image.
  
  - **shape**: (n_x, n_y, n_z), tuple of integers, optional
    
    The shape of the grid used for sampling, if None is given, a default affine is provided by the image.
  
  - **interpolation**: None, ‘continuous’ or ‘nearest’, optional
    
    Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.
  
  **Returns**
  
  - **resampled_image**: nipy VolumeImg
    
    New nipy VolumeImg with the data sampled on the grid defined by the affine and shape.

- `composed_with_transform(w2w_transform)`: Return a new image embedding the same data in a different word space using the given world to world transform.

- `get_data()`: Return data as a numpy array.

- `get_transform()`: Returns the transform object associated with the volumetric structure which is a general description of the mapping from the values to the world space.

- `like_from_data(data)`: Returns an volumetric data structure with the same relationship between data and world space, and same metadata, but different data.

- `resampled_to_img(target_image[, interpolation])`: Resample the data to be on the same voxel grid than the target volume structure.

- `values_in_world(x, y, z[, interpolation])`: Return the values of the data at the world-space positions given by

__init__()

x.__init__(...) initializes x; see help(type(x)) for signature

```
as_volume_img(affine=None, shape=None, interpolation=None, copy=True)
  Resample the image to be an image with the data points lying on a regular grid with an
  affine mapping to the word space (a nipy VolumeImg).
```

**Parameters**

- **affine**: 4x4 or 3x3 ndarray, optional
  
  Affine of the new voxel grid or transform object pointing to the new voxel coordinate
  grid. If a 3x3 ndarray is given, it is considered to be the rotation part of the affine, and
  the best possible bounding box is calculated, in this case, the shape argument is not
  used. If None is given, a default affine is provided by the image.

- **shape**: (n_x, n_y, n_z), tuple of integers, optional
  
  The shape of the grid used for sampling, if None is given, a default affine is provided
  by the image.

- **interpolation**: None, ‘continuous’ or ‘nearest’, optional
  
  Interpolation type used when calculating values in different word spaces. If None, the
  image’s interpolation logic is used.

**Returns**

- **resampled_image**: nipy VolumeImg
  
  New nipy VolumeImg with the data sampled on the grid defined by the affine and shape.

Notes

The coordinate system of the image is not changed: the returned image points to the same world space.

```
composed_with_transform(w2w_transform)
  Return a new image embedding the same data in a different word space using the given world to world transform.
```
**Parameters**  `w2w_transform` : transform object

The transform object giving the mapping between the current world space of the image, and the new world space.

**Returns**  `remapped_image` : nipy image

An image containing the same data, expressed in the new world space.

**get_data()**

Return data as a numpy array.

**get_transform()**

Returns the transform object associated with the volumetric structure which is a general description of the mapping from the values to the world space.

**interpolation = ‘continuous’**

**like_from_data(data)**

Returns an volumetric data structure with the same relationship between data and world space, and same metadata, but different data.

**Parameters**  `data`: ndarray :

**metadata = {}**

**resampled_to_img(target_image, interpolation=None)**

Resample the data to be on the same voxel grid than the target volume structure.

**Parameters**  `target_image` : nipy image

Nipy image onto the voxel grid of which the data will be resampled. This can be any kind of img understood by Nipy (datasets, pynifti objects, nibabel object) or a string giving the path to a nifti of analyse image.

**interpolation** : None, ‘continuous’ or ‘nearest’, optional

Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

**Returns**  `resampled_image` : nipy_image

New nipy image with the data resampled.

**Notes**

Both the target image and the original image should be embedded in the same world space.

**values_in_world(x, y, z, interpolation=None)**

Return the values of the data at the world-space positions given by x, y, z

**Parameters**  `x`: number or ndarray

x positions in world space, in other words millimeters

`y`: number or ndarray

y positions in world space, in other words millimeters. The shape of y should match the shape of x

`z`: number or ndarray
z positions in world space, in other words millimeters. The shape of z should match the shape of x

**interpolation** : None, ‘continuous’ or ‘nearest’, optional

Interpolation type used when calculating values in different word spaces. If None, the image's interpolation logic is used.

**Returns**  **values** : number or ndarray

Data values interpolated at the given world position. This is a number or an ndarray, depending on the shape of the input coordinate.

```python
world_space = ''
```
LABS.DATASETS.VOLUMES.VOLUME_FIELD

88.1 Module: labs.datasets.volumes.volume_field

Inheritance diagram for nipy.labs.datasets.volumes.volume_field:

```
volumes.volume_field.VolumeField
```

The base volumetric field interface
This defines the nipy volumetric structure interface.

88.2 VolumeField

```python
class nipy.labs.datasets.volumes.volume_field.VolumeField
    Bases: object

    The base volumetric structure.
    This object represents numerical values embedded in a 3-dimensional world space (called a field in physics and engineering)
    This is an abstract base class: it defines the interface, but not the logics.
```

Attributes

<table>
<thead>
<tr>
<th>world_space: string</th>
<th>World space the data is embedded in. For instance mni152.</th>
</tr>
</thead>
<tbody>
<tr>
<td>metadata: dictionary</td>
<td>Optional, user-defined, dictionary used to carry around extra information about the data as it goes through transformations. The consistency of this information is not maintained as the data is modified.</td>
</tr>
</tbody>
</table>
Methods

`as_volume_img([affine, shape, ...])`  
Resample the image to be an image with the data points lying

`composed_with_transform(w2w_transform)`  
Return a new image embedding the same data in a different word space using

`get_transform()`  
Returns the transform object associated with the volumetric structure which is a general description of the mapping from the values to the world space.

`resampled_to_img(target_image[, interpolation])`  
Resample the volume to be sampled similarly than the target volumetric structure.

`values_in_world(x, y, z[, interpolation])`  
Return the values of the data at the world-space positions given by

```
__init__()  
x.__init__(...) initializes x; see help(type(x)) for signature

as_volume_img (affine=None, shape=None, interpolation=None, copy=True)  
Resample the image to be an image with the data points lying on a regular grid with an affine mapping to the word space (a nipy VolumeImg).

Parameters  
affine: 4x4 or 3x3 ndarray, optional:
    Affine of the new voxel grid or transform object pointing to the new voxel coordinate grid. If a 3x3 ndarray is given, it is considered to be the rotation part of the affine, and the best possible bounding box is calculated, in this case, the shape argument is not used. If None is given, a default affine is provided by the image.

shape: (n_x, n_y, n_z), tuple of integers, optional:
    The shape of the grid used for sampling, if None is given, a default affine is provided by the image.

interpolation : None, ‘continuous’ or ‘nearest’, optional
    Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

Returns  
resampled_image : nipy VolumeImg
    New nipy VolumeImg with the data sampled on the grid defined by the affine and shape.

Notes
The coordinate system of the image is not changed: the returned image points to the same world space.

composed_with_transform(w2w_transform)
Return a new image embedding the same data in a different word space using the given world to world transform.

Parameters  
w2w_transform : transform object
    The transform object giving the mapping between the current world space of the image, and the new word space.

Returns  
remapped_image : nipy image
    An image containing the same data, expressed in the new world space.

get_transform()
Returns the transform object associated with the volumetric structure which is a general description of the mapping from the values to the world space.

Returns  
transform : nipy.datasets.Transform object

metadata = {}
resampled_to_img(target_image, interpolation=None)
Resample the volume to be sampled similarly than the target volumetric structure.

Parameters target_image: nipy volume
Nipy volume structure onto the grid of which the data will be resampled.

interpolation: None, ‘continuous’ or ‘nearest’, optional
Interpolation type used when calculating values in different word spaces. If None, the volume’s interpolation logic is used.

Returns resampled_image: nipy_image
New nipy image with the data resampled.

Notes
Both the target image and the original image should be embedded in the same world space.

values_in_world(x, y, z, interpolation=None)
Return the values of the data at the world-space positions given by x, y, z

Parameters x: number or ndarray
x positions in world space, in other words milimeters

y: number or ndarray
y positions in world space, in other words milimeters. The shape of y should match the shape of x

z: number or ndarray
z positions in world space, in other words milimeters. The shape of z should match the shape of x

interpolation: None, ‘continuous’ or ‘nearest’, optional
Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

Returns values: number or ndarray
Data values interpolated at the given world position. This is a number or an ndarray, depending on the shape of the input coordinate.
89.1 Module: labs.datasets.volumes.volume_grid

The volume grid class.
This class represents data lying on a (non rigid, non regular) grid embedded in a 3D world represented as a 3+D array.

89.2 VolumeGrid

class nipy.labs.datasets.volumes.volume_grid.VolumeGrid(data, transform, meta-
data=None, interpolation='continuous')

A class representing data stored in a 3+D array embedded in a 3D world.
This object has data stored in an array-like multidimensional indexable objects, with the 3 first dimensions corresponding to spatial axis and defining a 3D grid that may be non-regular or non-rigid.
The object knows how the data is mapped to a 3D “real-world space”, and how it can change real-world coordinate system. The transform mapping it to world is arbitrary, and thus the grid can be warped: in the world space, the grid may not be regular or orthogonal.

Notes

The data is stored in an undefined way: prescalings might need to be applied to it before using it, or the data might be loaded on demand. The best practice to access the data is not to access the _data attribute, but to use the get_data method.
If the transform associated with the image has no inverse mapping, data corresponding to a given world space position cannot be calculated. If it has no forward mapping, it is impossible to resample another dataset on the same support.
Attributes

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>world_space</td>
<td>World space the data is embedded in. For instance <em>mni152</em>.</td>
</tr>
<tr>
<td>metadata</td>
<td>Optional, user-defined dictionary used to carry around extra information about the data as it goes through transformations. The consistency of this information is not maintained as the data is modified.</td>
</tr>
<tr>
<td><em>data</em></td>
<td>Private pointer to the data.</td>
</tr>
</tbody>
</table>

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>as_volume_img([affine, shape, ...])</td>
<td>Resample the image to be an image with the data points lying on a regular grid with an affine mapping to the world space (a nipy VolumeImg).</td>
</tr>
<tr>
<td>composed_with_transform(w2w_transform)</td>
<td>Return a new image embedding the same data in a different world space using the given world to world transform.</td>
</tr>
<tr>
<td>get_data()</td>
<td>Return data as a numpy array.</td>
</tr>
<tr>
<td>get_transform()</td>
<td>Returns the transform object associated with the volumetric structure which is a general description of the mapping from the values to the world space.</td>
</tr>
<tr>
<td>get_world_coords()</td>
<td>Return the data points coordinates in the world space.</td>
</tr>
<tr>
<td>like_from_data(data)</td>
<td>Returns a volumetric data structure with the same relationship between data and world space, and same metadata, but different data.</td>
</tr>
<tr>
<td>resampled_to_img(target_image[, interpolation])</td>
<td>Resample the data to be on the same voxel grid than the target volume structure.</td>
</tr>
<tr>
<td>values_in_world(x, y, z[, interpolation])</td>
<td>Return the values of the data at the world-space positions given by the provided coordinates.</td>
</tr>
</tbody>
</table>

__init__ (data, transform, metadata=None, interpolation='continuous')

The base image containing data.

Parameters

- **data**: ndarray
  
n dimensional array giving the embedded data, with the 3 first dimensions being spatial.

- **transform**: nipy transform object
  
The transformation from voxel to world.

- **metadata**: dictionary, optional
  
A dictionary of user-specified information to store with the image.

- **interpolation**: ‘continuous’ or ‘nearest’, optional
  
Interpolation type used when calculating values in different word spaces.

as_volume_img (affine=None, shape=None, interpolation=None, copy=True)

Resample the image to be an image with the data points lying on a regular grid with an affine mapping to the word space (a nipy VolumeImg).

Parameters

- **affine**: 4x4 or 3x3 ndarray, optional
  
Affine of the new voxel grid or transform object pointing to the new voxel coordinate grid. If a 3x3 ndarray is given, it is considered to be the rotation part of the affine, and the best possible bounding box is calculated, in this case, the shape argument is not used. If None is given, a default affine is provided by the image.

- **shape**: (n_x, n_y, n_z), tuple of integers, optional
  
The shape of the grid used for sampling. If None is given, a default affine is provided by the image.

- **interpolation**: None, ‘continuous’ or ‘nearest’, optional
Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

**Returns** `resampled_image`: `nipy VolumeImg`

New `nipy VolumeImg` with the data sampled on the grid defined by the affine and shape.

**Notes**

The coordinate system of the image is not changed: the returned image points to the same world space.

`composed_with_transform(w2w_transform)`

Return a new image embedding the same data in a different word space using the given world to world transform.

**Parameters** `w2w_transform`: transform object

The transform object giving the mapping between the current world space of the image, and the new word space.

**Returns** `remapped_image`: `nipy image`

An image containing the same data, expressed in the new world space.

`get_data()`

Return data as a numpy array.

`get_transform()`

Returns the transform object associated with the volumetric structure which is a general description of the mapping from the values to the world space.

**Returns** `transform`: `nipy.datasets.Transform object`

`get_world_coords()`

Return the data points coordinates in the world space.

**Returns** `x`: `ndarray`

x coordinates of the data points in world space

`y`: `ndarray`

y coordinates of the data points in world space

`z`: `ndarray`

z coordinates of the data points in world space

**interpolation** = ‘continuous’

`like_from_data(data)`

Returns an volumetric data structure with the same relationship between data and world space, and same metadata, but different data.

**Parameters** `data`: `ndarray`

`metadata = {}`

`resampled_to_img(target_image, interpolation=None)`

Resample the data to be on the same voxel grid than the target volume structure.

**Parameters** `target_image`: `nipy image`
Nipy image onto the voxel grid of which the data will be resampled. This can be any kind of img understood by Nipy (datasets, pynifti objects, nibabel object) or a string giving the path to a nifti of analyse image.

**interpolation**: None, ‘continuous’ or ‘nearest’, optional

Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

**Returns** *resampled_image*: nipy_image

New nipy image with the data resampled.

**Notes**

Both the target image and the original image should be embedded in the same world space.

**values_in_world** *(x, y, z, interpolation=None)*

Return the values of the data at the world-space positions given by x, y, z

**Parameters** *x*: number or ndarray

x positions in world space, in other words millimeters

*y*: number or ndarray

y positions in world space, in other words millimeters. The shape of y should match the shape of x

*z*: number or ndarray

z positions in world space, in other words millimeters. The shape of z should match the shape of x

**interpolation**: None, ‘continuous’ or ‘nearest’, optional

Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

**Returns** *values*: number or ndarray

Data values interpolated at the given world position. This is a number or an ndarray, depending on the shape of the input coordinate.
90.1 Module: labs.datasets.volumes.volume_img

Inheritance diagram for nipy.labs.datasets.volumes.volume_img:

An image that stores the data as an \((x, y, z, \ldots)\) array, with an affine mapping to the world space

90.2 VolumeImg

class nipy.labs.datasets.volumes.volume_img.VolumeImg(data, affine, world_space, metadata=None, interpolation='continuous')

Bases: nipy.labs.datasets.volumes.volume_grid.VolumeGrid

A regularly-spaced image for embedding data in an \(x, y, z\) 3D world, for neuroimaging.

This object is an ndarray representing a volume, with the first 3 dimensions being spatial, and mapped to a named world space using an affine (4x4 matrix).

Notes

The data is stored in an undefined way: prescalings might need to be applied to it before using it, or the data might be loaded on demand. The best practice to access the data is not to access the \_data attribute, but to use the get_data method.

Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>affine</td>
<td>str(obj) -&gt; string</td>
</tr>
<tr>
<td>world_space</td>
<td>str(obj) -&gt; string</td>
</tr>
<tr>
<td>metadata</td>
<td>Continued on next page</td>
</tr>
</tbody>
</table>

Continued on next page
Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>_data :</td>
<td>Private pointer to the data.</td>
</tr>
<tr>
<td>as_volume_img(</td>
<td>affine, shape, ...</td>
</tr>
<tr>
<td>composed_with_transform(w2w_transform)</td>
<td>Return a new image embedding the same data in a different word space using</td>
</tr>
<tr>
<td>get_affine()</td>
<td>Returns the transform object associated with the volumetric structure which is a general description of the mapping from the values to the world space.</td>
</tr>
<tr>
<td>get_data()</td>
<td>Return data as a numpy array.</td>
</tr>
<tr>
<td>get_world_coords()</td>
<td>Return the data points coordinates in the world space.</td>
</tr>
<tr>
<td>like_from_data(data)</td>
<td>Returns a volumetric data structure with the same relationship between data and the world space, and the same metadata, but different data.</td>
</tr>
<tr>
<td>resampled_to_img(target_image[, interpolation])</td>
<td>Resample the data to be on the same voxel grid than the target volume structure.</td>
</tr>
<tr>
<td>values_in_world(x, y, z[, interpolation])</td>
<td>Return the values of the data at the world-space positions given by</td>
</tr>
<tr>
<td>xyz_ordered([resample, copy])</td>
<td>Returns an image with the affine diagonal and positive in the world space it is embedded in.</td>
</tr>
</tbody>
</table>

__init__(data, affine, world_space, metadata=None, interpolation='continuous')

Creates a new neuroimaging image with an affine mapping.

**Parameters**

- **data**: ndarray
  ndarray representing the data.
- **affine**: 4x4 ndarray
  affine transformation to the reference world space
- **world_space**: string
  name of the reference world space.
- **metadata**: dictionary
  dictionary of user-specified information to store with the image.
- **affine**: 4x4 or 3x3 ndarray, optional:
  Affine of the new voxel grid or transform object pointing to the new voxel coordinate grid. If a 3x3 ndarray is given, it is considered to be the rotation part of the affine, and the best possible bounding box is calculated, in this case, the shape argument is not used. If None is given, a default affine is provided by the image.
- **shape**: (n_x, n_y, n_z), tuple of integers, optional:
  The shape of the grid used for sampling. If None is given, a default affine is provided by the image.
- **interpolation**: None, ‘continuous’ or ‘nearest’, optional
Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

**Returns**  
`resampled_image : nipy VolumeImg`

New nipy VolumeImg with the data sampled on the grid defined by the affine and shape.

**Notes**

The coordinate system of the image is not changed: the returned image points to the same world space.

**composed_with_transform**(`w2w_transform`)  
Return a new image embedding the same data in a different word space using the given world to world transform.

**Parameters**  
`w2w_transform : transform object`

The transform object giving the mapping between the current world space of the image, and the new word space.

**Returns**  
`remapped_image : nipy image`

An image containing the same data, expressed in the new world space.

**get_affine**()

**get_data**()  
Return data as a numpy array.

**get_transform**()  
Returns the transform object associated with the volumetric structure which is a general description of the mapping from the values to the world space.

**Returns**  
`transform : nibabel.datasets.Transform object`

**get_world_coords**()  
Return the data points coordinates in the world space.

**Returns**  
`x : ndarray`

x coordinates of the data points in world space

`y : ndarray`

y coordinates of the data points in world space

`z : ndarray`

z coordinates of the data points in world space

**interpolation** = 'continuous'

**like_from_data**(`data`)  
Returns an volumetric data structure with the same relationship between data and world space, and same metadata, but different data.

**Parameters**  
`data : ndarray`

`metadata = {}`

**resampled_to_img**(`target_image, interpolation=None`)  
Resample the data to be on the same voxel grid than the target volume structure.

**Parameters**  
`target_image : nipy image`
Nipy image onto the voxel grid of which the data will be resampled. This can be any kind of img understood by Nipy (datasets, pynifti objects, nibabel object) or a string giving the path to a nifti of analyse image.

**interpolation** : None, ‘continuous’ or ‘nearest’, optional

Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

**Returns** resampled_image : nipy_image

New nipy image with the data resampled.

**Notes**

Both the target image and the original image should be embedded in the same world space.

**values_in_world** (x, y, z, interpolation=None)

Return the values of the data at the world-space positions given by `x`, `y`, `z`.

**Parameters**

- `x` : number or ndarray
  
  x positions in world space, in other words millimeters

- `y` : number or ndarray
  
  y positions in world space, in other words millimeters. The shape of `y` should match the shape of `x`.

- `z` : number or ndarray
  
  z positions in world space, in other words millimeters. The shape of `z` should match the shape of `x`.

- `interpolation` : None, ‘continuous’ or ‘nearest’, optional
  
  Interpolation type used when calculating values in different word spaces. If None, the image’s interpolation logic is used.

**Returns** values : number or ndarray

Data values interpolated at the given world position. This is a number or an ndarray, depending on the shape of the input coordinate.

**world_space** = ‘’

**xyz_ordered** (resample=False, copy=True)

Returns an image with the affine diagonal and positive in the world space it is embedded in.

**Parameters**

- `resample` : boolean, optional
  
  If resample is False, no resampling is performed, the axis are only permuted. If it is impossible to get xyz ordering by permuting the axis, a ‘CompositionError’ is raised.

- `copy` : boolean, optional
  
  If copy is True, a deep copy of the image (including the data) is made.
LABS.GLMBENCHMARKSBENCH_GLM

91.1 Module: labs.glm.benchmarks.bench_glm

91.2 Functions

nipy.labs.glm.benchmarks.bench_glm.bench_ols_axis0()
nipy.labs.glm.benchmarks.bench_glm.bench_ols_axis1()
nipy.labs.glm.benchmarks.bench_glm.bench_ols_axis2()
nipy.labs.glm.benchmarks.bench_glm.bench_ols_axis3()
nipy.labs.glm.benchmarks.bench_glm.make_data()
nipy.labs.glm.benchmarks.bench_glm.ols(ax, y, X)
92.1 Module: labs.glm.glm

Inheritance diagram for nipy.labs.glm.glm:

```python
glm.glm.contrast

92.2 Classes

92.2.1 contrast

class nipy.labs.glm.glm.contrast (dim, type='t', tiny=1e-50, dofmax=10000000000.0)
Bases: object

Methods

    pvalue([baseline])  Return a parametric approximation of the p-value associated
                      with the test of the contrast.

    stat([baseline])    Return the decision statistic associated with the test of the
                      contrast.

    summary()          Return a dictionary containing the estimated contrast effect,
                      zscore([baseline])  Return a parametric approximation of the z-score associated
                      with the test of the contrast.

    __init__ (dim, type='t', tiny=1e-50, dofmax=10000000000.0)
                      tiny is a numerical constant for computations.

    pvalue (baseline=0.0)
                      Return a parametric approximation of the p-value associated with the null hypothesis:
                      (H0) 'contrast equals 0'.
```
baseline'

**stat (baseline=0.0)**

Return the decision statistic associated with the test of the null hypothesis: (H0) ‘contrast equals baseline’

**summary ()**

Return a dictionary containing the estimated contrast effect, the associated ReML-based estimation variance, and the estimated degrees of freedom (variance of the variance).

**zscore (baseline=0.0)**

Return a parametric approximation of the z-score associated with the null hypothesis: (H0) ‘contrast equals baseline’

### 92.2.2 glm

**class nipy.labs.glm.glm (Y=None, X=None, formula=None, axis=0, model='spherical', method=None, niter=2)**

* Bases: object

**Methods**

- **contrast(c[, type, tiny, dofmax])** Specify and estimate a constrast c must be a numpy.ndarray (or anything that numpy.asarray can cast to a ndarray).
- **fit(Y, X[, formula, axis, model, method, niter])**
- **save(file)** Save fit into a .npz file

**__init__ (Y=None, X=None, formula=None, axis=0, model='spherical', method=None, niter=2)**

**contrast (c, type='t', tiny=1e-50, dofmax=10000000000.0)** Specify and estimate a constrast
c must be a numpy.ndarray (or anything that numpy.asarray can cast to a ndarray). For a F contrast, c must be q x p where q is the number of contrast vectors and p is the total number of regressors.

**fit (Y, X, formula=None, axis=0, model='spherical', method=None, niter=2)**

**save (file)** Save fit into a .npz file

### 92.3 Functions

- **nipy.labs.glm.glm.load (file)** Load a fitted glm
- **nipy.labs.glm.glm.ols (Y, X, axis=0)** Essentially, compute pinv(X)*Y
93.1 Module: labs.group.displacement_field

Inheritance diagram for nipy.labs.group.displacement_field:

```
<table>
<thead>
<tr>
<th>group.displacement_field.gaussian_random_field</th>
</tr>
</thead>
<tbody>
<tr>
<td>group.displacement_field.displacement_field</td>
</tr>
</tbody>
</table>
```

93.2 Classes

93.2.1 displacement_field

```python
class nipy.labs.group.displacement_field.displacement_field(XYZ, sigma, n=1, mask=None, step=None):
    Bases: object
    Sampling of multiple vector-valued displacement fields on a 3D-lattice. Displacement fields are generated as linear combinations of fixed displacements. The coefficients are random Gaussian variables.
```

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>compute_inner_blocks()</td>
<td>Generate self.inner_blocks, index of blocks which are “far from” the borders</td>
</tr>
<tr>
<td>init_displacement_blocks()</td>
<td>Called by class constructor</td>
</tr>
<tr>
<td>sample(i, b[, proposal, proposal_std, ...])</td>
<td>Generates U, V, L, W, I, where U, V, W, I are proposals for</td>
</tr>
<tr>
<td>sample_all_blocks([proposal_std, proposal_mean])</td>
<td>Generates U, V, W, I, proposals for self.U[:, i], self.V[:, i], self.W[:, i], self.I[:, i]</td>
</tr>
</tbody>
</table>
__init__(XYZ, sigma, n=1, mask=None, step=None)

Input: XYZ (3,p) array of voxel coordinates sigma <float> standard deviate of Gaussian filter kernel

Each displacement block has length 4*sigma

n <int> number of generated displacement fields. mask (q,) displacement blocks are limited to mask

The constructor creates the following fields:
- self.block List of N block masks (voxel index vectors)
- self.weights List of N block weights (same shape as the masks)
- self.U (3,n,N) Displacement coefficients
- self.V (3,n,p) Displacements
- self.W (3,n,p) Discretize displacements
- self.I (n,p) Displaced voxels index

 voxel k in the mask is displaced by field i to voxel self.I[i,k]

compute_inner_blocks()

Generate self.inner_blocks, index of blocks which are “far from” the borders of the lattice.

init_displacement_blocks()

Called by class constructor

sample(i, b, proposal='prior', proposal_std=None, proposal_mean=None)

Generates U, V, W, I, where U, V, W, I are proposals for self.U[:,i,b], self.V[:,i,b], self.W[:,i,L],

self.I[i,L] if block = self.block[b]. W and I are given only in those voxels, indexed by L, where they differ

from current values. Proposal is either ‘prior’, ‘rand_walk’ or ‘fixed’

sample_all_blocks(proposal_std=None, proposal_mean=None)


‘rand_walk’ or ‘fixed’

93.2.2 gaussian_random_field

class nipy.labs.group.displacement_field.gaussian_random_field(XYZ, sigma, n=1)

Bases: object

Methods

sample(i, std)

__init__(XYZ, sigma, n=1)

sample(i, std)

93.3 Functions

nipy.labs.group.displacement_field.square_gaussian_filter(input, sigma,

output=None, mode='reflect', cval=0.0)

Multi-dimensional Squared Gaussian filter.

The standard-deviations of the Gaussian filter are given for each axis as a sequence, or as a single number, in

which case it is equal for all axes.

Note: The multi-dimensional filter is implemented as a sequence of one-dimensional convolution filters. The

intermediate arrays are stored in the same data type as the output. Therefore, for output types with a limited

precision, the results may be imprecise because intermediate results may be stored with insufficient precision.
nipy.labs.group.displacement_field.square_gaussian_filter1d(input, sigma, axis=-1, output=None, mode='reflect', cval=0.0)

One-dimensional Squared Gaussian filter.

The standard-deviation of the Gaussian filter is given by sigma.
LABS.GROUP.MIXED_EFFECTS

94.1 Module: labs.group.mixed_effects

New generic implementation of multiple regression analysis under noisy measurements.

94.2 Functions

nipy.labs.group.mixed_effects.em(Y, VY, X=None, niter=2, log_likelihood=False)
Maximum likelihood regression in a mixed-effect linear model using the EM algorithm.

Parameters
Y : array
Array of observations.

VY : array
C is the contrast matrix. Conventionally, C is p x q where p:

is the number of regressors.

OUTPUT: beta, s2:

beta – array of parameter estimates:
s2 – array of squared scale parameters:

REFERENCE:
Keller and Roche, ISBI 2008.

nipy.labs.group.mixed_effects.log_likelihood_ratio(Y, VY, X, C, niter=2)
Log-likelihood ratio statistic: 2*(log L - log L0)
It is asymptotically distributed like a chi-square with rank(C) degrees of freedom under the null hypothesis H0: Cb = 0.

nipy.labs.group.mixed_effects.nonzero(x)
Force strictly positive values.
CHAPTER
NINETYFIVE

LABS.GROUPPERMUTATION_TEST

95.1 Module: labs.group.permutation_test

Inheritance diagram for nipy.labs.group.permutation_test:

One and two sample permutation tests.

95.2 Classes

95.2.1 permutation_test

class nipy.labs.group.permutation_test.permutation_test
    Bases: object

    This generic permutation test class contains the calibration method which is common to the derived classes
    permutation_test_onesample and permutation_test_twosample (as well as other common methods)

    Methods

    | method               | Description                                           |
    |----------------------|-------------------------------------------------------|
    | calibrate ([nperms, clusters, cluster_stats, ...]) | Calibrate cluster and region summary statistics using permutation test |
    | height_threshold(pval)  | Return the uniform height threshold matching a given p-value |
    | pvalue([Tvalues])      | Return uncorrected voxel-level pseudo p-values.      |
    | zscore([Tvalues])      | Return z score corresponding to the uncorrected      |
__init__()
x.__init__(...) initializes x; see help(type(x)) for signature

calibrate(nperms=10000, clusters=None, cluster_stats=['size', 'Fisher'], regions=None, region_stats=['Fisher'], verbose=False)
Calibrate cluster and region summary statistics using permutation test

Parameters

nperms : int, optional
Number of random permutations generated. Exhaustive permutations are used only if nperms=None, or exceeds total number of possible permutations

clusters : list [(thresh1,diam1),(thresh2,diam2),...], optional
List of cluster extraction pairs: (thresh,diam). thresh provides T values threshold, diam is the maximum cluster diameter, in voxels. Using *diam*==None yields classical suprathreshold clusters.

cluster_stats : list [stat1,...], optional
List of cluster summary statistics id (either ‘size’ or ‘Fisher’)

regions : list [Labels1,Labels2,...]
List of region labels arrays, of size (p,) where p is the number of voxels

region_stats : list [stat1,...], optional
List of cluster summary statistics id (only ‘Fisher’ supported for now)

verbose : boolean, optional
“Chatterbox” mode switch

Returns

voxel_results : dict
A dictionary containing the following keys: p_values (p,) Uncorrected p-values. “Corr_p_values” (p,) Corrected p-values, computed by the Tmax procedure. perm_maxT_values (nperms) values of the maximum statistic under permutation.

cluster_results : list [results1,results2,...]
List of permutation test results for each cluster extraction pair. These are dictionaries with the following keys “thresh”, “diam”, “labels”, “expected_voxels_per_cluster”, “expected_number_of_clusters”, and “peak_XYZ” if XYZ field is nonempty and for each summary statistic id “S”: “size_values”, “size_p_values”, “S_Corr_p_values”, “perm_size_values”, “perm_maxsize_values”

region_results : list [results1,results2,...]
List of permutation test results for each region labels arrays. These are dictionaries with the following keys: “label_values”, “peak_XYZ” (if XYZ field nonempty) and for each summary statistic id “S”: “size_values”, “size_p_values”, “perm_size_values”, “perm_maxsize_values”

height_threshold(pval)
Return the uniform height threshold matching a given permutation-based P-value.

pvalue(Tvalues=None)
Return uncorrected voxel-level pseudo p-values.

zscore(Tvalues=None)
Return z score corresponding to the uncorrected voxel-level pseudo p-value.
95.2.2 permutation_test_onesample

class nipy.labs.group.permutation_test.permutation_test_onesample:

Bases: nipy.labs.group.permutation_test.permutation_test

Class derived from the generic permutation_test class. Inherits the calibrate method

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>calibrate</td>
<td>Calibrate cluster and region summary statistics using permutation test</td>
</tr>
<tr>
<td>height_threshold</td>
<td>Return the uniform height threshold matching a given pvalue</td>
</tr>
<tr>
<td>pvalue</td>
<td>Return uncorrected voxel-level pseudo p-values.</td>
</tr>
<tr>
<td>zscore</td>
<td>Return z score corresponding to the uncorrected</td>
</tr>
</tbody>
</table>

__init__:

Initialize permutation_test_onesample instance, compute statistic values in each voxel and under permutation
In: data data array

XYZ voxels coordinates axis <int> Subject axis in data
vardata variance (same shape as data) optional (if None, mfx statistics cannot be used)
stat_id <char> choice of test statistic (see onesample.stats for a list of possible stats)
base <float> mean signal under H0 niter <int> number of iterations of EM algorithm ndraws
<int> Number of generated random t values

Out: self.Tvalues voxelwise test statistic values self.random_Tvalues sorted statistic values in random
voxels and under random

sign permutation

calibrate (nperms=10000, clusters=None, cluster_stats=['size', 'Fisher'], regions=None, region_stats=['Fisher'], verbose=False)

Calibrate cluster and region summary statistics using permutation test

Parameters:

nperms : int, optional
Number of random permutations generated. Exhaustive permutations are used only if
nperms=None, or exceeds total number of possible permutations
clusters : list [(thresh1,diam1),(thresh2,diam2),...], optional
List of cluster extraction pairs: (thresh,diam). thresh provides T values threshold, diam
is the maximum cluster diameter, in voxels. Using *diam*=None yields classical
suprathreshold clusters.
cluster_stats : list [stat1,...], optional
List of cluster summary statistics id (either 'size' or 'Fisher')
regions : list [Labels1,Labels2,...]
List of region labels arrays, of size (p,) where p is the number of voxels

**region_stats** : list [stat1,...], optional
List of cluster summary statistics id (only ‘Fisher’ supported for now)

**verbose** : boolean, optional
“Chatterbox” mode switch

**Returns voxel_results**: dict
A dictionary containing the following keys: p_values (p,) Uncorrected p-values, “Corr_p_values” (p,) Corrected p-values, computed by the Tmax procedure. perm_maxT_values (nperms) values of the maximum statistic under permutation.

**cluster_results**: list [results1,results2,...]
List of permutation test results for each cluster extraction pair. These are dictionaries with the following keys: “thresh”, “diam”, “labels”, “expected_voxels_per_cluster”, “expected_number_of_clusters”, and “peak_XYZ” if XYZ field is nonempty and for each summary statistic id “S”: “size_values”, “size_p_values”, “S_Corr_p_values”, “perm_size_values”, “perm_maxsize_values”

**region_results**: list [results1,results2,...]
List of permutation test results for each region labels arrays. These are dictionaries with the following keys: “label_values”, “peak_XYZ” (if XYZ field nonempty) and for each summary statistic id “S”: “size_values”, “size_p_values”, “perm_size_values”, “perm_maxsize_values”

**height_threshold** (pval)
Return the uniform height threshold matching a given permutation-based P-value.

**pvalue** (Tvalues=None)
Return uncorrected voxel-level pseudo p-values.

**zscore** (Tvalues=None)
Return z score corresponding to the uncorrected voxel-level pseudo p-value.

### 95.2.3 permutation_test_onesample_graph

class nipy.labs.group.permutation_test.permutation_test_onesample_graph (data, G, axis=0, var_data=None, stat_id='student', base=0.0, niter=5, ndraws=100000)

Bases: nipy.labs.group.permutation_test.permutation_test

Class derived from the generic permutation_test class. Inherits the calibrate method

**Methods**

**calibrate**([nperms, clusters, cluster_stats, ...]) Calibrate cluster and region summary statistics using permutation test

Continued on next page
Table 95.3 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>height_threshold(pval)</code></td>
<td>Return the uniform height threshold matching a given p-value.</td>
</tr>
<tr>
<td><code>pvalue([Tvalues])</code></td>
<td>Return uncorrected voxel-level pseudo p-values.</td>
</tr>
<tr>
<td><code>zscore([Tvalues])</code></td>
<td>Return z score corresponding to the uncorrected T values.</td>
</tr>
</tbody>
</table>

`__init__`(data, G, axis=0, vardata=None, stat_id='student', base=0.0, niter=5, ndraws=100000)

Initialize permutation_test_onesample instance, compute statistic values in each voxel and under permutation.

In: data data array
G weighted graph (each vertex corresponds to a voxel) axis <int> Subject axis in data vardata variance (same shape as data)
optional (if None, mfx statistics cannot be used)

stat_id <char> choice of test statistic (see onesample.stats for a list of possible stats)
base <float> mean signal under H0 niter <int> number of iterations of EM algorithm ndraws <int> Number of generated random t values

Out: self.Tvalues voxelwise test statistic values self.random_Tvalues sorted statistic values in random voxels and under random sign permutation

calibrate(nperms=10000, clusters=None, cluster_stats=['size', 'Fisher'], regions=None, region_stats=['Fisher'], verbose=False)

Calibrate cluster and region summary statistics using permutation test

Parameters |
--- |
**nperms** : int, optional
Number of random permutations generated. Exhaustive permutations are used only if nperms=None, or exceeds total number of possible permutations

**clusters** : list [(thresh1,diam1),(thresh2,diam2)...], optional
List of cluster extraction pairs: (thresh,diam). thresh provides T values threshold, diam is the maximum cluster diameter, in voxels. Using *diam*=None yields classical suprathreshold clusters.

**cluster_stats** : list [stat1,...], optional
List of cluster summary statistics id (either 'size' or 'Fisher')

**regions** : list [Labels1,Labels2,...]
List of region labels arrays, of size (p,) where p is the number of voxels

**region_stats** : list [stat1,...], optional
List of cluster summary statistics id (only 'Fisher' supported for now)

**verbose** : boolean, optional
“Chatterbox” mode switch

Returns |
--- |
**voxel_results** : dict
A dictionary containing the following keys: p_values (p,) Uncorrected p-values, ‘Corr_p_values’ (p,) Corrected p-values, computed by the Tmax procedure.
perm_maxT_values (nperms) values of the maximum statistic under permutation.

**cluster_results** : list [results1,results2,...]
List of permutation test results for each cluster extraction pair. These are dictionaries with the following keys “thresh”, “diam”, “labels”, “expected_voxels_per_cluster”, “expected_number_of_clusters”, and “peak_XYZ” if XYZ field is nonempty and for each summary statistic id “S”: “size_values”, “size_p_values”, “S_Corr_p_values”, “perm_size_values”, “perm_maxsize_values”

region_results : list [results1, results2, ...]

List of permutation test results for each region labels arrays. These are dictionaries with the following keys: “label_values”, “peak_XYZ” (if XYZ field nonempty) and for each summary statistic id “S”: “size_values”, “size_p_values”, “perm_size_values”, “perm_maxsize_values”

height_threshold (pval)
Return the uniform height threshold matching a given permutation-based P-value.

pvalue (Tvalues=None)
Return uncorrected voxel-level pseudo p-values.

zscore (Tvalues=None)
Return z score corresponding to the uncorrected voxel-level pseudo p-value.

95.2.4 permutation_test_twosample

class nipy.labs.group.permutation_test.permutation_test_twosample (data1, data2, XYZ, axis=0, vardata1=None, vardata2=None, stat_id='student', niter=5, ndraws=100000)

Bases: nipy.labs.group.permutation_test.permutation_test

Class derived from the generic permutation_test class. Inherits the calibrate method

Methods

calibrate([nperms, clusters, cluster_stats, ...]) Calibrate cluster and region summary statistics using permutation test

height_threshold(pval) Return the uniform height threshold matching a given

pvalue([Tvalues]) Return uncorrected voxel-level pseudo p-values.

zscore([Tvalues]) Return z score corresponding to the uncorrected

__init__ (data1, data2, XYZ, axis=0, vardata1=None, vardata2=None, stat_id='student', niter=5, ndraws=100000)

Initialize permutation_test_twosample instance, compute statistic values in each voxel and under permutation In: data1, data2 data arrays

XYZ voxels coordinates axis <int> Subject axis in data

vardata1, vardata2 variance (same shape as data) optional (if None, mfx statistics cannot be used)

stat_id <char> choice of test statistic (see onesample.stats for a list of possible stats)
niter <int> number of iterations of EM algorithm ndraws <int> Number of generated random t values

Out: self.Tvalues voxelwise test statistic values self.random_Tvalues sorted statistic values in random voxels and under random sign permutation

calibrate (nperms=10000, clusters=None, cluster_stats=[‘size’, ‘Fisher’], regions=None, region_stats=[‘Fisher’], verbose=False)
Calibrate cluster and region summary statistics using permutation test

Parameters

  nperms : int, optional
    Number of random permutations generated. Exhaustive permutations are used only if nperms=None, or exceeds total number of possible permutations

  clusters : list [(thresh1,diam1),(thresh2,diam2),...], optional
    List of cluster extraction pairs: (thresh,diam). thresh provides T values threshold, diam is the maximum cluster diameter, in voxels. Using *diam*==None yields classical suprathreshold clusters.

  cluster_stats : list [stat1,...], optional
    List of cluster summary statistics id (either 'size' or 'Fisher')

  regions : list [Labels1,Labels2,...]
    List of region labels arrays, of size (p,) where p is the number of voxels

  region_stats : list [stat1,...], optional
    List of cluster summary statistics id (only 'Fisher' supported for now)

  verbose : boolean, optional
    “Chatterbox” mode switch

Returns

  voxel_results : dict
    A dictionary containing the following keys: p_values (p,) Uncorrected p-values. ‘Corr_p_values’ (p,) Corrected p-values, computed by the Tmax procedure.

  perm_maxT_values (nperms) values of the maximum statistic under permutation.

  cluster_results : list [results1,results2,...]
    List of permutation test results for each cluster extraction pair. These are dictionaries with the following keys “thresh”, “diam”, “labels”, “expected_voxels_per_cluster”, “expected_number_of_clusters”, and “peak_XYZ” if XYZ field is nonempty and for each summary statistic id “S”: “size_values”, “size_p_values”, “S_Corr_p_values”, “perm_size_values”, “perm_maxsize_values”

  region_results : list [results1,results2,...]
    List of permutation test results for each region labels arrays. These are dictionaries with the following keys: “label_values”, “peak_XYZ” (if XYZ field nonempty) and for each summary statistic id “S”: “size_values”, “size_p_values”, “perm_size_values”, “perm_maxsize_values”

height_threshold (pval)
    Return the uniform height threshold matching a given permutation-based P-value.

pvalue (Tvalues=None)
    Return uncorrected voxel-level pseudo p-values.
\texttt{zscore (Tvalues=None)}

Return z score corresponding to the uncorrected voxel-level pseudo p-value.

### 95.3 Functions

\texttt{nipy.labs.group.permutation_test.compute_cluster_stats (Tvalues, labels, random_Tvalues, cluster_stats=['size', 'Fisher'])}

\texttt{size_values, Fisher_values = compute_cluster_stats(Tvalues, labels, random_Tvalues, cluster_stats=['size', 'Fisher'])} Compute summary statistics in each cluster In: see permutation_test_onesample class docstring Out: size_values Array of size nclust, or None if “size” not in cluster_stats

\texttt{Fisher_values Array of size nclust, or None if “Fisher” not in cluster_stats}

\texttt{nipy.labs.group.permutation_test.compute_region_stat (Tvalues, labels, label_values, random_Tvalues)}

\texttt{Fisher_values = compute_region_stat(Tvalues, labels, label_values, random_Tvalues)} Compute summary statistics in each cluster In: see permutation_test_onesample class docstring Out: Fisher_values Array of size nregions

\texttt{nipy.labs.group.permutation_test.extract_clusters_from_diam (T, XYZ, th, diam, k=18)}

Extract clusters from a statistical map under diameter constraint and above given threshold In: T (p) statistical map

\texttt{XYZ (3,p) voxels coordinates th <float> minimum threshold diam <int> maximal diameter (in voxels) k <int> the number of neighbours considered. (6,18 or 26)}

Out: labels (p) cluster labels

Comment by alexis-roche, September 15th 2012: this function was originally developed by Merlin Keller in an attempt to generalize classical cluster-level analysis by subdividing clusters in blobs with limited diameter (at least, this is my understanding). This piece of code seems to have remained very experimental and its usefulness in real-world neuroimaging image studies is still to be demonstrated.

\texttt{nipy.labs.group.permutation_test.extract_clusters_from_graph (T, G, th)}

This returns a label vector of same size as T, defining connected components for subgraph of weighted graph G containing vertices s.t. T >= th

\texttt{nipy.labs.group.permutation_test.extract_clusters_from_thresh (T, XYZ, th, k=18)}

Extract clusters from statistical map above specified threshold In: T (p) statistical map

\texttt{XYZ (3,p) voxels coordinates th <float> threshold k <int> the number of neighbours considered. (6,18 or 26)}

Out: labels (p) cluster labels

\texttt{nipy.labs.group.permutation_test.max_dist (XYZ, I, J)}

Maximum distance between two set of points In: XY (3,p) voxels coordinates

\texttt{I (q) index of points J (r) index of points}

Out: d <float>

\texttt{nipy.labs.group.permutation_test.onesample_stat (Y, V, stat_id, base=0.0, axis=0, Magic=None, niter=5)}

Wrapper for os_stat and os_stat_mfx

\texttt{nipy.labs.group.permutation_test.peak_XYZ (XYZ, Tvalues, labels, label_values)}

Returns (3, n_labels) array of maximum T values coordinates for each label value
nipy.labs.group.permutation_test.sorted_values(a)

Extract list of distinct sorted values from an array

nipy.labs.group.permutation_test.twosample_stat(Y1, V1, Y2, V2, stat_id, axis=0, Mag-
ciales=0, niter=5)

Wrapper for ts_stat and ts_stat_mfx
LABS.GROUP.SPATIAL_RELAXATION_ONESAMPLE

96.1 Module: labs.group.spatial_relaxation_onesample

Inheritance diagram for nipy.labs.group.spatial_relaxation_onesample:

```
  group.spatial_relaxation_onesample.multivariate_stat
```
96.2 Class

96.3 multivariate_stat

class nipy.labs.group.spatial_relaxation_onsample.multivariate_stat:

data, var-
data=None, 
XYZ=None, 
std=None, 
sigma=None, 
la-
bel=None, 
network=None, 
v.shape=3, 
v.scale=20, 
std.shape=3, 
std.scale=20, 
m.mean.rate=0.001, 
m.var.shape=3, 
m.var.scale=20, 
disp_mask=None, 
la-
bel_prior=None, 
la-
bel_values=None, 
la-
bel_prior_mask=None)

Bases: object

Methods

compute_conditional_posterior_mean([v, ...]) Compute posterior mean of mean effect map.
compute_log_conditional_displacements_posterior([...]) Compute posterior log density of elementary displacements.
compute_log_conditional_posterior([v, ...]) Compute log posterior density of model parameters, conditional on hidden parameters.
compute_log_posterior([v, m_mean, m_var, ...]) Compute log posterior density of region parameters by Rao-Blackwell method.
compute_log_prior([v, m_mean, m_var, std]) Compute log prior density of model parameters, spatial uncertainty excepted.
compute_log_region_likelihood([v, m_mean, m_var]) Compute log region likelihood.
compute_log_region_likelihood_slow([v, ...]) Essentially maintained for debug purposes.
compute_log_voxel_likelihood([v, m_mean, ...]) Compute log voxel likelihood.
compute_marginal_likelihood([v, m_mean, ...]) Compute marginal likelihood.
estimate_displacements_SA([nsimu, c, ...]) MAP estimate of elementary displacements conditional on model parameters.
evaluate([nsimu, burnin, J, verbose, ...]) Sample posterior distribution of model parameters, or compute log posterior.
init_hidden_variables([mode, init.spatial]) Initialize hidden variables.
sample_log_conditional_posterior([v, ...]) Sample log conditional posterior density of region parameters.
update_block(i, b[, proposal, proposal_std, ...]) Update displacement block.
update_block_SA(i, b[, T, proposal_std, ...]) Update displacement block using simulated annealing.
update_displacements() Update displacements.
update_displacements_SA([T, proposal_std, ...]) Update displacements.
update_effects([T]) T is a temperature used to compute log posterior density.
**update_labels()**

**update_mean_effect([T])**  
*T* is a temperature used to compute log posterior density

**update_parameters_mcmc(update_spatial)**

**update_parameters_saem(update_spatial)**

**update_summary_statistics([w, ...])**

---

**__init__**(data, vardata=None, XYZ=None, std=None, sigma=None, labels=None, network=None, v_shape=3, v_scale=20, std_shape=3, std_scale=20, m_mean_rate=0.001, m_var_shape=3, m_var_scale=20, disp_mask=None, labels_prior=None, label_values=None, labels_prior_mask=None)

Multivariate modeling of fMRI group data accounting for spatial uncertainty In: data (n,p) estimated effects

vardata (n,p) variances of estimated effects XYZ (3,p) voxel coordinates std <float> Initial guess for standard deviate of spatial displacements sigma <float> regularity of displacement field labels (p,) labels defining regions of interest network (N,) binary region labels (1 for active, 0 for inactive) v_shape <float> intensity variance prior shape v_scale <float> intensity variance prior scale std_shape <float> spatial standard error prior shape std_scale <float> spatial standard error prior scale m_mean_rate <float> mean effect prior rate m_var_shape <float> effect variance prior shape m_var_scale <float> effect variance prior scale disp_mask (q,) mask of the brain, to limit displacements labels_prior (M,r) prior on voxelwise region membership labels_prior_values (M,r) voxelwise label values where prior is defined labels_prior_mask (r,) Mask of voxels where a label prior is defined

**compute_conditional_posterior_mean**(v=None, m_mean=None, m_var=None)

Compute posterior mean of mean effect map, conditional on parameters and displacements

**compute_log_conditional_displacements_posterior**(U=None, nsimu=100, burnin=100, proposal_std=None, verbose=False, change_U=False)

Compute posterior log density of elementary displacements at point U, conditional on model parameters

**compute_log_conditional_posterior**(v=None, m_mean=None, m_var=None, std=None)

compute log posterior density of model parameters, conditional on hidden parameters. This function is used in compute_log_region_posterior. It should only be used within the Gibbs sampler, and not the SAEM algorithm.

**compute_log_posterior**(v=None, m_mean=None, m_var=None, std=None, nsimu=100, burnin=100, stabilize=False, verbose=False, update_spatial=False)

compute log posterior density of region parameters by Rao-Blackwell method, or a stabilized upper bound if stabilize is True.

**compute_log_prior**(v=None, m_mean=None, m_var=None, std=None)

calculate log prior density of model parameters, spatial uncertainty excepted, assuming hidden variables have been initialized

**compute_log_region_likelihood**(v=None, m_mean=None, m_var=None)

**compute_log_region_likelihood_slow**(v=None, m_mean=None, m_var=None, verbose=False, J=None)

Essentially maintained for debug purposes

**compute_log_voxel_likelihood**(v=None, m_mean=None, m_var=None, return_SS=False)

**compute_marginal_likelihood**(v=None, m_mean=None, m_var=None, std=None, nsimu=100, burnin=100, stabilize=False, verbose=False, update_spatial=False, U=None, proposal_std=None)

---

96.3. multivariate_stat 647
**estimate_displacements_SA** *(nsimu=100, c=0.99, proposal_std=None, verbose=False)*

MAP estimate of elementary displacements conditional on model parameters

**evaluate** *(nsimu=1000.0, burnin=100, J=None, verbose=False, proposal=’prior’, proposal_std=None, proposal_mean=None, compute_post_mean=False, mode=’saem’, update_spatial=True)*

Sample posterior distribution of model parameters, or compute their MAP estimator In: nsimu <int>
Number of samples drawn from posterior mean distribution

burnin <int> Number of discarded burn-in samples J (N,) voxel indices where successive mean values are stored verbose <bool> Print some infos during the sampling process proposal <str> ‘prior’, ‘rand_walk’ or ‘fixed’ proposal_mean <float> Used for fixed proposal only proposal_std <float> Used for random walk or fixed proposal mode <str> if mode=’saem’, compute MAP estimates of model parameters.

if mode=’mcmc’, sample their posterior distribution

update_spatial <bool> when False, enables sampling conditional on spatial parameters

Out: self.m_values (N, nsimu+burnin) successive mean values (if J is not empty) if self.labels_priors is not empty:

self.labels_post (M,r) posterior distribution of region labels

if self.std is not empty: self.std_values (nsimu+burnin,) successive spatial standard deviate values

if compute_post_mean is True: self.mean_m (p,) posterior average of mean effect self.var_m (p,) posterior variance of mean effect

if self.std is not empty and compute_post_mean is True: self.r (n, nblocs) mean rejection rate for each displacement field self.mean_U (3, n, nblocs) posterior average of displacement weights self.var_U (3, n, nblocs) posterior marginal variances of displacement weights

**init_hidden_variables** *(mode=’saem’, init_spatial=True)*

**sample_log_conditional_posterior** *(v=None, m_mean=None, m_var=None, std=None, nsimu=100, burnin=100, stabilize=False, verbose=False, update_spatial=False)*

Sample log conditional posterior density of region parameters using a Gibbs sampler (assuming all hidden variables have been initialized). Computes posterior mean. if stabilize is True, sampling is conditioned on the parameters, reducing the variance of the estimate, but introducing a positive bias.

**update_block** *(i, b, proposal=’prior’, proposal_std=None, proposal_mean=None, verbose=False, reject_override=False)*

**update_block_SA** *(i, b, T=1.0, proposal_std=None, proposal_mean=None, proposal=’rand_walk’, proposal_mean=None)*

Update displacement block using simulated annealing scheme with random-walk kernel

**update_displacements** ()

**update_displacements_SA** *(T=1.0, proposal_std=None, verbose=False)*

**update_effects** *(T=1.0)*

T is a temperature used to compute log posterior density by simulated annealing

**update_labels** ()

**update_mean_effect** *(T=1.0)*

T is a temperature used to compute log posterior density by simulated annealing

**update_parameters_mcmc** *(update_spatial=True)*

**update_parameters_saem** *(update_spatial=True)*
update_summary_statistics(w=1.0, update_spatial=True, mode='saem')

96.4 Functions

nipy.labs.group.spatial_relaxation.onesample.log_gammainv_pdf(x, a, b)
   log density of the inverse gamma distribution with shape a and scale b, at point x, using Stirling’s approximation for a > 100

nipy.labs.group.spatial_relaxation.onesample.log_gaussian_pdf(x, m, v)
   log density of the gaussian distribution with mean m and variance v at point x
97.1 Module: labs.mask

Utilities for extracting masks from EPI images and applying them to time series.

97.2 Functions

nipy.labs.mask.compute_mask(\textit{mean\_volume}, \textit{reference\_volume}=\text{None}, m=0.2, M=0.9, \textit{cc}=\text{True},
          \textit{opening}=2, \textit{exclude\_zeros}=\text{False})

Compute a mask file from fMRI data in 3D or 4D ndarrays.

Compute and write the mask of an image based on the grey level This is based on an heuristic proposed by T.Nichols: find the least dense point of the histogram, between fractions \( m \) and \( M \) of the total image histogram.

In case of failure, it is usually advisable to increase \( m \).

Parameters  \textit{mean\_volume} : 3D ndarray

mean EPI image, used to compute the threshold for the mask.

\textit{reference\_volume} : 3D ndarray, optional :

reference volume used to compute the mask. If none is give, the mean volume is used.

\textit{m} : float, optional

lower fraction of the histogram to be discarded.

\textit{M} : float, optional :

upper fraction of the histogram to be discarded.

\textit{cc} : boolean, optional :

if \textit{cc} is \text{True}, only the largest connect component is kept.

\textit{opening} : int, optional :

if \textit{opening} is larger than 0, an morphological opening is performed, to keep only large structures. This step is useful to remove parts of the skull that might have been included.

\textit{exclude\_zeros} : boolean, optional :

Consider zeros as missing values for the computation of the threshold. This option is useful if the images have been resliced with a large padding of zeros.

Returns  \textit{mask} : 3D boolean ndarray
Compute a mask file from fMRI nifti file(s)

**Parameters**

- **input_filename**: string
  - nifti filename (4D) or list of filenames (3D).

- **output_filename**: string or None, optional
  - path to save the output nifti image (if not None).

- **return_mean**: boolean, optional
  - if True, and output_filename is None, return the mean image also, as a 3D array (2nd return argument).

- **m**: float, optional
  - lower fraction of the histogram to be discarded.

- **M**: float, optional
  - upper fraction of the histogram to be discarded.

- **cc**: boolean, optional
  - if cc is True, only the largest connect component is kept.

- **exclude_zeros**: boolean, optional
  - Consider zeros as missing values for the computation of the threshold. This option is useful if the images have been resliced with a large padding of zeros.

- **opening**: int, optional
  - Size of the morphological opening performed as post-processing

**Returns**

- **mask**: 3D boolean array
  - The brain mask

- **mean_image**: 3D ndarray, optional
  - The main of all the images used to estimate the mask. Only provided if return_mean is True.

---

Compute a common mask for several sessions of fMRI data.

**Parameters**

- **session_images**: list of (list of strings) or nipy image objects
  - A list of images/list of nifti filenames. Each inner list/image represents a session.

- **m**: float, optional
lower fraction of the histogram to be discarded.

M: float, optional :
upper fraction of the histogram to be discarded.

c: boolean, optional :
if cc is True, only the largest connect component is kept.

threshold : float, optional
the inter-session threshold: the fraction of the total number of session in for which a voxel must be in the mask to be kept in the common mask. threshold=1 corresponds to keeping the intersection of all masks, whereas threshold=0 is the union of all masks.

exclude_zeros: boolean, optional :
Consider zeros as missing values for the computation of the threshold. This option is useful if the images have been resliced with a large padding of zeros.

return_mean: boolean, optional :
if return_mean is True, the mean image accross subjects is returned.

opening: int, optional,
size of the morphological opening

Returns mask : 3D boolean ndarray
The brain mask

mean : 3D float array
The mean image

nipy.labs.mask.intersect_masks (input_masks, output_filename=None, threshold=0.5, cc=True)
Given a list of input mask images, generate the output image which is the the threshold-level intersection of the inputs

Parameters input_masks: list of strings or ndarrays :
paths of the input images nsubj set as len(input_mask_files), or individual masks.

output_filename, string: :
Path of the output image, if None no file is saved.

threshold: float within [0, 1[, optional :
gives the level of the intersection. threshold=1 corresponds to keeping the intersection of all masks, whereas threshold=0 is the union of all masks.

c: bool, optional :
If true, extract the main connected component

Returns grp_mask, boolean array of shape the image shape :

nipy.labs.mask.largest_cc (mask)
Return the largest connected component of a 3D mask array.

Parameters mask: 3D boolean array :
3D array indicating a mask.

Returns mask: 3D boolean array :
3D array indicating a mask, with only one connected component.

```python
nipy.labs.mask.series_from_mask(filenames, mask, dtype='numpy.float32', smooth=False, ensure_finite=True)
```

Read the time series from the given sessions filenames, using the mask.

**Parameters**

- **filenames**: list of 3D nifti file names, or 4D nifti filename.
  
  Files are grouped by session.

- **mask**: 3d ndarray
  
  3D mask array: true where a voxel should be used.

- **smooth**: False or float, optional
  
  If smooth is not False, it gives the size, in voxel of the spatial smoothing to apply to the signal.

- **ensure_finite**: boolean, optional
  
  If ensure_finite is True, the non-finite values (NaNs and infs) found in the images will be replaced by zeros

**Returns**

- **session_series**: ndarray
  
  3D array of time course: (session, voxel, time)

- **header**: header object
  
  The header of the first file.

**Notes**

When using smoothing, ensure_finite should be True: as elsewhere non finite values will spread across the image.

```python
nipy.labs.mask.threshold_connect_components(map, threshold, copy=True)
```

Given a map with some coefficients set to zero, segment the connect components with number of voxels smaller than the threshold and set them to 0.

**Parameters**

- **map**: ndarray
  
  The spatial map to segment

- **threshold**: scalar
  
  The minimum number of voxels to keep a cluster.

- **copy**: bool, optional
  
  If copy is false, the input array is modified inplace

**Returns**

- **map**: ndarray
  
  the map with connected components removed
LABS.SPATIAL_MODELS.BAYESIAN_STRUCTURAL_ANALYSIS

98.1 Module: labs.spatial_models.bayesian_structural_analysis

The main routine of this package that aims at performing the extraction of ROIs from multisubject dataset using the localization and activation strength of extracted regions.

This has been published in: - Thirion et al. High level group analysis of FMRI data based on Dirichlet process mixture models, IPMI 2007 - Thirion et al. Accurate Definition of Brain Regions Position Through the Functional Landmark Approach, MICCAI 2010

Author : Bertrand Thirion, 2006-2011

98.2 Functions

nipy.labs.spatial_models.bayesian_structural_analysis.bsa_dpmm(bf, gf0, sub, gfc, dmax, thq, ths, verbose=0)

Estimation of the population level model of activation density using dpmm and inference

Parameters

- **bf** list of nipy.labs.spatial_models.hroi.HierarchicalROI instances:
  representing individual ROIs let nr be the number of terminal regions across subjects
- **gf0**, array of shape (nr):
  the mixture-based prior probability that the terminal regions are true positives
- **sub**, array of shape (nr):
  the subject index associated with the terminal regions
- **gfc**, array of shape (nr, coord.shape[1]):
  the coordinates of the of the terminal regions
- **dmax** float>0:
  expected cluster std in the common space in units of coord
- **thq = 0.5 (float in the [0,1] interval)**:
  p-value of the prevalence test
- **ths=0, float in the range [0,nsubj]**:
  null hypothesis on region prevalence that is rejected during inference
verbose=0, verbosity mode:

Returns crmap: array of shape (nnodes):
the resulting group-level labelling of the space

LR: a instance of sbf.LandmarkRegions that describes the ROIs found:
in inter-subject inference If no such thing can be defined LR is set to None

bf: List of nipy.labs.spatial_models.hroi.Nroi instances:
representing individual ROIs

p: array of shape (nnodes):
likelihood of the data under H1 over some sampling grid

nipy.labs.spatial_models.bayesian_structural_analysis.bsa_dpmm2(bf, gf0, sub, gfc, dmax, thq, ths, verbose)

Estimation of the population level model of activation density using dpmm and inference

Parameters

bf list of nipy.labs.spatial_models.hroi.HierarchicalROI instances:
representing individual ROIs let nr be the number of terminal regions across subjects

gf0, array of shape (nr):
the mixture-based prior probability that the terminal regions are false positives

sub, array of shape (nr):
the subject index associated with the terminal regions

gfc, array of shape (nr, coord.shape[1]):
the coordinates of the of the terminal regions

dmax float>0:
expected cluster std in the common space in units of coord

thq = 0.5 (float in the [0,1] interval):
p-value of the prevalence test

ths=0, float in the range [0,nsubj]:
null hypothesis on region prevalence that is rejected during inference

verbose=0, verbosity mode:

Returns crmap: array of shape (nnodes):
the resulting group-level labelling of the space

LR: a instance of sbf.LandmarkRegions that describes the ROIs found:
in inter-subject inference If no such thing can be defined LR is set to None

bf: List of nipy.labs.spatial_models.hroi.Nroi instances:
representing individual ROIs

Coclust: array of shape (nr,nr):
co-labelling matrix that gives for each pair of inputs how likely they are in the same class according to the model
Compute the Bayesian Structural Activation patterns - with statistical validation

**Parameters**
- **dom**: StructuredDomain instance,
  - Description of the spatial context of the data
- **lbeta**: an array of shape (nbnodes, subjects),
  - the multi-subject statistical maps
- **dmax** float>0,
  - expected cluster std in the common space in units of coord
- **thq** = 0.5 (float),
  - posterior significance threshold should be in the [0,1] interval
- **smin** = 5 (int),
  - minimal size of the regions to validate them
- **ths** = 0 (float),
  - first level threshold
- **method**: string, optional,
  - the method used to assess the prior significance of the regions
- **verbose**=0: verbosity mode

**Returns**
- **mll** : float
  - the average cross-validated log-likelihood across subjects
- **ml0** : float
  - the log-likelihood of the model under a global null hypothesis

Idem `compute_BSA_simple`, but this one does not estimate the full density (on small datasets, it can be much faster)

**Parameters**
- **dom**: StructuredDomain instance,
  - Description of the spatial context of the data
- **lbeta**: an array of shape (nbnodes, subjects),
  - the multi-subject statistical maps
- **dmax** float>0,
Compute the Bayesian Structural Activation patterns simplified version

**Parameters**

- **dom**: StructuredDomain instance,
  
  Description of the spatial context of the data

- **lbeta**: an array of shape (nbnodes, subjects):
  
  the multi-subject statistical maps

- **dmax float>0**: 
  
  expected cluster std in the common space in units of coord

- **thq = 0.5 (float)**: 
  
  posterior significance threshold should be in the [0,1] interval

- **smin = 5 (int)**: minimal size of the regions to validate them:

- **theta = 3.0 (float)**: first level threshold:

- **method**: string, optional,
  
  the method used to assess the prior significance of the regions

- **verbose=0**: verbosity mode:

**Returns**

- **crmap**: array of shape (nnodes):
  
  the resulting group-level labelling of the space

- **LR**: a instance of sbf.LandmarkRegions that describes the ROIs found:
  
  in inter-subject inference If no such thing can be defined LR is set to None

- **bf**: List of nipy.labs.spatial_models.hroi.Nroi instances:
  
  representing individual ROIs

- **coclust**: array of shape (nr, nr):
  
  co-labelling matrix that gives for each pair of cross_subject regions how likely they are in the same class according to the model

```python
nipy.labs.spatial_models.bayesian_structural_analysis.compute_BSA_simple(dom, lbeta, dmax, thq=0.5, smin=5, ths=0, theta=3.0, method='prior', verbose=0)
```
verbose=0: verbosity mode:

Returns:

crmap: array of shape (nnodes):

the resulting group-level labelling of the space

LR: a instance of sbf.LandmarkRegions that describes the ROIs found:

in inter-subject inference If no such thing can be defined LR is set to None

bf: List of nipy.labs.spatial_models.hroi.Nroi instances:

representing individual ROIs

p: array of shape (nnodes):

likelihood of the data under H1 over some sampling grid

Notes

In that case, the DPMM is used to derive a spatial density of significant local maxima in the volume. Each terminal (leaf) region which is a posteriori significant enough is assigned to the nearest mode of this distribution

nipy.labs.spatial_models.bayesian_structural_analysis.compute_individual_regions(domain, lbeta, smin=5, theta=3.0, method='gauss_mixture', verbose=0, reshuffle=0, criterion='size', assign_val='weighted_mean')

Compute the individual regions that are real activation candidates

Parameters:

dom: StructuredDomain instance,

generic descriptor of the space domain

lbeta: an array of shape (nbnodes, subjects):

the multi-subject statistical maps

smin: int, optional:

minimal size of the regions to validate them

theta: float, optional:

first level threshold

method: string, optional:

method that is used to provide priori significance can be ‘prior’, ‘gauss_mixture’, ‘gam_gauss’ or ‘emp_null’

verbose: verbosity mode, optional:

reshuffle: bool, optional,
if nonzero, reshuffle the positions; this affects bf and gfc

criterion: string, optional, :
    ‘size’ or ‘volume’, thresholding criterion

assign_val: string, optional, :
    to be chosen in ‘weighted mean’, ‘mean’, ‘min’, ‘max’ heuristic to assign a blob-level signal

Returns  bf list of nipy.labs.spatial_models.hroi.Nroi instances :
    representing individual ROIs let nr be the number of terminal regions across subjects
gf0, array of shape (nr) :
    the mixture-based prior probability that the terminal regions are false positives
sub, array of shape (nr) :
    the subject index associated with the terminal regions
gfc, array of shape (nr, coord.shape[1]) :
    the coordinates of the of the terminal regions

Apply the dpmm analysis to compute clusters from regions coordinates

Convert a set of z-values to posterior probabilities of not being active

Parameters  test: array pf shape(n_samples), :
    data that is assessed

learn: array pf shape(n_samples), optional :
    data to learn a mixture model

method: string, optional, to be chosen within :
    [‘gauss_mixture’, ‘emp_null’, ‘gam_gauss’, ‘prior’]

alpha: float in the [0,1], optional, :
    parameter that yields the prior probability that a region is active should be chosen close to 0
99.1 Module: labs.spatial_models.bsa_io

This module is the interface to the bayesian_structural_analysis (bsa) module. It handles the images provided as input and produces result images.

```python
def make_bsa_image(mask_images, betas, theta=3.0, dmax=5.0, ths=0, thq=0.5, smin=0, swd=None, method='simple', subj_id=None, nbeta='default', dens_path=None, cr_path=None, verbose=0, resuffle=False)
```

Main function for performing bsa on a set of images. It creates some output images in the given directory.

**Parameters**

- `mask_images`: A list of image paths that yield binary images, one for each subject. The number of subjects, nsubj, is taken as len(mask_images).
- `betas`: A list of image paths that yield activation images, one for each subject.
- `theta=3.0`, threshold used to ignore all the image data that is below.
- `dmax=5.0`, prior width of the spatial model. corresponds to multi-subject uncertainty.
- `ths=0`: threshold on the representativity measure of the obtained regions.
- `thq=0.5`: p-value of the representativity test:
  - test = p(reprensentativity>ths)>thq
- `smin=0`: minimal size (in voxels) of the extracted blobs:
  - smaller blobs are merged into larger ones
- `swd`: string, optional:
  - if not None, output directory
- `method='simple'`: applied region detection method; to be chose among:
  - 'simple', 'quick', 'loo'
- `subj_id=None`: list of strings, identifiers of the subjects.
by default it is range(nsubj)

**nbeta='default', string, identifier of the contrast:**

**dens_path=None, string, path of the output density image:**

if False, no image is written if None, the path is computed from swd, nbeta

**cr_path=None, string, path of the (4D) output label image:**

if False, no image is written if None, many images are written, with paths computed from swd, subj_id and nbeta

**reshuffle: bool, optional:**

if true, randomly swap the sign of the data

**Returns**

**AF:** an nipy.labs.spatial_models.structural_bfls.landmark_regions instance that describes the structures found at the group level
None is returned if nothing has been found significant at the group level

**BF:** a list of nipy.labs.spatial_models.hroi.Nroi instances
(one per subject) that describe the individual counterpart of AF

**if method==’loo’, the output is different:**

mll, float, the average likelihood of the data under the model after cross validation
ll0, float the log-likelihood of the data under the global null
100.1 Module: labs.spatial_models.discrete_domain

Inheritance diagram for nipy.labs.spatial_models.discrete_domain:

```
spatial_models.discrete_domain.MeshDomain
spatial_models.discrete_domain.DiscreteDomain
spatial_models.discrete_domain.StructuredDomain
spatial_models.discrete_domain.NDGridDomain
```

This module defines the StructuredDomain class, that represents a generic neuroimaging kind of domain This is meant to provide a unified API to deal with n-d imaged and meshes.

Author: Bertrand Thirion, 2010

100.2 Classes

100.2.1 DiscreteDomain

```python
class nipy.labs.spatial_models.discrete_domain.DiscreteDomain(dim, coord, local_volume, id='', referential='')
```

Bases: object

Descriptor of a certain domain that consists of discrete elements that are characterized by a coordinate system and a topology: the coordinate system is specified through a coordinate array the topology encodes the neighboring system

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>connected_components()</code></td>
<td>returns a labelling of the domain into connected components</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Returns a copy of self</td>
</tr>
<tr>
<td><code>get_coord()</code></td>
<td>Returns self.coord</td>
</tr>
<tr>
<td><code>get_feature(fid)</code></td>
<td>Return self.features[fid]</td>
</tr>
<tr>
<td><code>get_volume()</code></td>
<td>Returns self.local_volume</td>
</tr>
</tbody>
</table>

Continued on next page
**Table 100.1 – continued from previous page**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>integrate(fid)</code></td>
<td>Integrate certain feature over the domain and returns the result</td>
</tr>
<tr>
<td><code>mask(bmask[, id])</code></td>
<td>Returns an DiscreteDomain instance that has been further masked</td>
</tr>
<tr>
<td><code>representative_feature(fid, method)</code></td>
<td>Compute a statistical representative of the within-Foain feature</td>
</tr>
<tr>
<td><code>set_feature(fid, data[, override])</code></td>
<td>Append a feature ‘fid’</td>
</tr>
</tbody>
</table>

```python
__init__(dim, coord, local_volume, id='', referential='')

Initialize discrete domain instance

Parameters

- **dim**: int, necessary
  - the (physical) dimension of the domain
- **coord**: array of shape(size, em_dim), necessary
  - explicit coordinates of the domain sites
- **local_volume**: array of shape(size), necessary
  - yields the volume associated with each site
- **id**: string, optional
  - domain identifier
- **referential**: string, optional
  - identifier of the referential of the coordinates system

`connected_components()`
returns a labelling of the domain into connected components

`copy()`
Returns a copy of self

`get Coord()`
Returns self.coord

`get_feature(fid)`
Return self.features[fid]

`get_volume()`
Returns self.local_volume

`integrate(fid)`
Integrate certain feature over the domain and returns the result

Parameters

- **fid**: string, feature identifier,
  - by default, the 1 function is integrated, yielding domain volume

Returns

- `lsum = array of shape (self.feature[fid].shape[1])`, necessary
  - the result

`mask(bmask, id='')`
Returns an DiscreteDomain instance that has been further masked

`representative_feature(fid, method)`
Compute a statistical representative of the within-Foain feature

Parameters

- **fid**: string, feature id
  - method: string, method used to compute a representative
    - to be chosen among ‘mean’, ‘max’, ‘median’, ‘min’
```

664 Chapter 100. labs.spatial_models.discrete_domain
**set_feature** (*fid, data, override=True*)
Append a feature ‘fid’

**Parameters**
- **fid**: string,
  feature identifier
- **data**: array of shape(self.size, p) or self.size,
  the feature data

### 100.2.2 MeshDomain

class `nipy.labs.spatial_models.discrete_domain.MeshDomain` (*coord, triangles*)

temporary class to handle meshes

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>area()</code></td>
<td>Return array of areas for each node</td>
</tr>
<tr>
<td><code>topology()</code></td>
<td>Returns a sparse matrix that represents the connectivity in self</td>
</tr>
</tbody>
</table>

**__init__** (*coord, triangles*)
Initialize mesh domain instance

**Parameters**
- **coord**: array of shape (*n_vertices, 3*),
  the node coordinates
- **triangles**: array of shape(*n_triables, 3*),
  indices of the nodes per triangle

**area**()
Return array of areas for each node

**Returns**
- **area**: array of shape `self.V`,
  area of each node

**topology**()
Returns a sparse matrix that represents the connectivity in self

### 100.2.3 NDGridDomain

class `nipy.labs.spatial_models.discrete_domain.NDGridDomain` (*dim, ijk, shape, affine, local_volume, topology, referential='')

Bases: `nipy.labs.spatial_models.discrete_domain.StructuredDomain`

Particular instance of StructuredDomain, that receives 3 additional variables: `affine`: array of shape (*dim+1, dim+1*),

- **affine**: affine transform that maps points to a coordinate system

- **shape**: `dim-tuple`, shape of the domain
**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>connected_components()</code></td>
<td>Returns a labelling of the domain into connected components</td>
</tr>
<tr>
<td><code>copy()</code></td>
<td>Returns a copy of self</td>
</tr>
<tr>
<td><code>get_coord()</code></td>
<td>Returns <code>self.coord</code></td>
</tr>
<tr>
<td><code>get_feature(fid)</code></td>
<td>Return <code>self.features[fid]</code></td>
</tr>
<tr>
<td><code>get_volume()</code></td>
<td>Returns <code>self.local_volume</code></td>
</tr>
<tr>
<td><code>integrate(fid)</code></td>
<td>Integrate certain feature over the domain and returns the result</td>
</tr>
<tr>
<td><code>make_feature_from_image(path[, fid])</code></td>
<td>Extract the information from an image to make it a domain a feature</td>
</tr>
<tr>
<td><code>mask(bmask)</code></td>
<td>Returns an instance of self that has been further masked</td>
</tr>
<tr>
<td><code>representative_feature(fid, method)</code></td>
<td>Compute a statistical representative of the within-Foain feature</td>
</tr>
<tr>
<td><code>set_feature(fid, data[, override])</code></td>
<td>Append a feature <code>fid</code></td>
</tr>
<tr>
<td><code>to_image([path, data])</code></td>
<td>Write itself as a binary image, and returns it</td>
</tr>
</tbody>
</table>

**__init__**(dim, ijk, shape, affine, local_volume, topology, referential='' )

Initialize ndgrid domain instance

**Parameters**
- **dim**: int.
  - the (physical) dimension of the domain
- **ijk**: array of shape(size, dim), int.
  - grid coordinates of the points
- **shape**: dim-tuple.
  - shape of the domain
- **affine**: array of shape (dim+1, dim+1),
  - affine transform that maps points to a coordinate system
- **local_volume**: array of shape(size),
  - yields the volume associated with each site
- **topology**: sparse binary coo_matrix of shape (size, size),
  - that yields the neighboring locations in the domain
- **referential**: string, optional,
  - identifier of the referential of the coordinates system

  `connected_components()`
  - returns a labelling of the domain into connected components

  `copy()`
  - Returns a copy of self

  `get_coord()`
  - Returns `self.coord`

  `get_feature(fid)`
  - Return `self.features[fid]`
get_volume()  
Returns self.local_volume

integrate(fid)  
Integrate certain feature over the domain and returns the result

Parameters  
fid : string, feature identifier,
        by default, the 1 function is integrated, yielding domain volume

Returns  
lsum = array of shape (self.feature[fid].shape[1]), :
        the result

make_feature_from_image(path, fid='')  
Extract the information from an image to make it a domain a feature

Parameters  
path: string or Nifti1Image instance, :
        the image from which one wished to extract data

fid: string, optional :
        identifier of the resulting feature, if '', the feature is not stored

Returns  
the corresponding set of values :

mask(bmask)  
Returns an instance of self that has been further masked

representative_feature(fid, method)  
Compute a statistical representative of the within-Foain feature

Parameters  
fid: string, feature id :

method: string, method used to compute a representative :
        to be chosen among ‘mean’, ‘max’, ‘median’, ‘min’

set_feature(fid, data, override=True)  
Append a feature ‘fid’

Parameters  
fid: string, :
        feature identifier

data: array of shape(self.size, p) or self.size :
        the feature data

to_image(path=None, data=None)  
Write itself as a binary image, and returns it

Parameters  
path: string, path of the output image, if any :

data: array of shape self.size, :
        data to put in the nonzer-region of the image

100.2.4 StructuredDomain

class nipy.labs.spatial_models.discrete_domain.StructuredDomain(dim, coord, 
local_volume, topology, did='', referential='')

Bases: nipy.labs.spatial_models.discrete_domain.DiscreteDomain
Besides DiscreteDomain attributed, StructuredDomain has a topology, which allows many operations (morphology etc.)

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>connected_components()</td>
<td>returns a labelling of the domain into connected components</td>
</tr>
<tr>
<td>copy()</td>
<td>Returns a copy of self</td>
</tr>
<tr>
<td>get_coord()</td>
<td>Returns self.coord</td>
</tr>
<tr>
<td>get_feature(fid)</td>
<td>Return self.features[fid]</td>
</tr>
<tr>
<td>get_volume()</td>
<td>Returns self.local_volume</td>
</tr>
<tr>
<td>integrate(fid)</td>
<td>Integrate certain feature over the domain and returns the result</td>
</tr>
<tr>
<td>mask(bmask[, did])</td>
<td>Returns a StructuredDomain instance that has been further masked</td>
</tr>
<tr>
<td>representative_feature(fid, method)</td>
<td>Compute a statistical representative of the within-Foain feature</td>
</tr>
<tr>
<td>set_feature(fid, data[, override])</td>
<td>Append a feature ‘fid’</td>
</tr>
</tbody>
</table>

```python
__init__(dim, coord, local_volume, topology, did='', referential='')
```

Initialize structured domain instance

**Parameters**

- **dim**: int,
  - the (physical) dimension of the domain

- **coord**: array of shape(size, em_dim),
  - explicit coordinates of the domain sites

- **local_volume**: array of shape(size),
  - yields the volume associated with each site

- **topology**: sparse binary coo_matrix of shape (size, size),
  - that yields the neighboring locations in the domain

- **did**: string, optional,
  - domain identifier

- **referential**: string, optional,
  - identifier of the referential of the coordinates system

- **connected_components()**
  - returns a labelling of the domain into connected components

- **copy()**
  - Returns a copy of self

- **get_coord()**
  - Returns self.coord

- **get_feature(fid)**
  - Return self.features[fid]

- **get_volume()**
  - Returns self.local_volume

- **integrate(fid)**
  - Integrate certain feature over the domain and returns the result

**Parameters**

- **fid**: string, feature identifier,
by default, the 1 function is integrataed, yielding domain volume

**Returns**  
`lsum = array of shape (self.feature[fid].shape[1]),`  
the result

**mask** *(bmask, did='')*

Returns a StructuredDomain instance that has been further masked

**representative_feature** *(fid, method)*

Compute a statistical representative of the within-Foain feature

**Parameters**  
**fid:** string, feature id

**method:** string, method used to compute a representative

- to be chosen among 'mean', 'max', 'median', 'min'

**set_feature** *(fid, data, override=True)*

Append a feature ‘fid’

**Parameters**  
**fid:** string,

**data:** array of shape(self.size, p) or self.size

the feature data

### 100.3 Functions

**nipy.labs.spatial_models.discrete_domain.array_affine_coord** *(mask, affine)*

Compute coordinates from a boolean array and an affine transform

**Parameters**  
**mask:** nd array,

input array, interpreted as a mask

**affine:** (n+1, n+1) matrix,

affine transform that maps the mask points to some embedding space

**Returns**  
`coords: array of shape(sum(mask>0), n),`  
the computed coordinates

**nipy.labs.spatial_models.discrete_domain.domain_from_binary_array** *(mask, affine=None, nn=0)*

Return a StructuredDomain from an n-d array

**Parameters**  
**mask:** np.array instance

a supposedly boolean array that represents the domain

**affine:** np.array, optional

affine transform that maps the array coordinates to some embedding space by default,  
this is np.eye(dim+1, dim+1)

**nn:** neighboring system considered

unsued at the moment
nipy.labs.spatial_models.discrete_domain.domain_from_image(mim, nn=18)

Return a StructuredDomain instance from the input mask image

Parameters  
mim: NiftiImage instance, or string path toward such an image:
  
supposedly a mask (where is used to crate the DD)

nn: int, optional:
  
neighboring system considered from the image can be 6, 18 or 26

Returns  
The corresponding StructuredDomain instance:

nipy.labs.spatial_models.discrete_domain.domain_from_mesh(mesh)

Instantiate a StructuredDomain from a gifti mesh

Parameters  
mesh: nibabel gifti mesh instance, or path to such a mesh:

nipy.labs.spatial_models.discrete_domain.grid_domain_from_binary_array(mask, 
  
affine=None, 
  
nn=0)

Return a NDGridDomain from an n-d array

Parameters  
mask: np.array instance:
  
a supposedly boolean array that represents the domain

affine: np.array, optional:
  
affine transform that maps the array coordinates to some embedding space by default, 
  
this is np.eye(dim+1, dim+1)

nn: neighboring system considered:
  
unused at the moment

nipy.labs.spatial_models.discrete_domain.grid_domain_from_image(mim, nn=18)

Return a NDGridDomain instance from the input mask image

Parameters  
mim: NiftiImage instance, or string path toward such an image:
  
supposedly a mask (where is used to crate the DD)

nn: int, optional:
  
neighboring system considered from the image can be 6, 18 or 26

Returns  
The corresponding NDGridDomain instance:

nipy.labs.spatial_models.discrete_domain.grid_domain_from_shape(shape, 
  
affine=None)

Return a NDGridDomain from an n-d array

Parameters  
shape: tuple:
  
the shape of a rectangular domain.

affine: np.array, optional:
  
affine transform that maps the array coordinates to some embedding space. By default, 
  
this is np.eye(dim+1, dim+1)

nipy.labs.spatial_models.discrete_domain.idx_affine_coord(idx, affine)

Compute coordinates from a set of indexes and an affine transform

Parameters  
idx: array of shape (n_samples, dim), type int:
  
indexes of certain positions in a nd space
**affine**: \((n+1, n+1)\) matrix, :

affine transform that maps the mask points to some embedding space

**Returns**

**coords**: array of shape\((\text{sum(mask}>0), n)\), :

the computed coordinates

```python
nipy.labs.spatial_models.discrete_domain.reduce_coo_matrix(mat, mask)
```

Reduce a supposedly coo_matrix to the vertices in the mask

**Parameters**

**mat**: sparse coo_matrix, :

input matrix

**mask**: boolean array of shape \(\text{mat.shape}[0]\), :

desired elements

```python
nipy.labs.spatial_models.discrete_domain.smatrix_from_3d_array(mask, nn=18)
```

Create a sparse adjacency matrix from an array

**Parameters**

**mask**: 3d array, :

input array, interpreted as a mask

**nn**: int, optional :

3d neighboring system to be chosen within \(\{6, 18, 26\}\)

**Returns**

**coo_mat**: a sparse coo_matrix, :

adjacency of the neighboring system

```python
nipy.labs.spatial_models.discrete_domain.smatrix_from_3d_idx(ijk, nn=18)
```

Create a sparse adjacency matrix from 3d index system

**Parameters**

**ijk**: array of shape \((\text{n_samples}, 3)\), type int :

indexes of certain positions in a 3d space

**nn**: int, optional :

3d neighboring system to be chosen within \(\{6, 18, 26\}\)

**Returns**

**coo_mat**: a sparse coo_matrix, :

adjacency of the neighboring system

```python
nipy.labs.spatial_models.discrete_domain.smatrix_from_nd_array(mask, nn=0)
```

Create a sparse adjacency matrix from an arbitrary nd array

**Parameters**

**mask**: nd array, :

input array, interpreted as a mask

**nn**: int, optional :

nd neighboring system, unused at the moment

**Returns**

**coo_mat**: a sparse coo_matrix, :

adjacency of the neighboring system

```python
nipy.labs.spatial_models.discrete_domain.smatrix_from_nd_idx(idx, nn=0)
```

Create a sparse adjacency matrix from nd index system

**Parameters**

**idx**: array of shape \((\text{n_samples}, \text{dim})\), type int :

indexes of certain positions in a nd space
nn: int, optional:

nd neighboring system, unused at the moment

Returns coo_mat: a sparse coo matrix:

adjacency of the neighboring system
101.1 Module: labs.spatial_models.hierarchical_parcellation

Computation of parcellations using a hierarchical approach. Author: Bertrand Thirion, 2008

101.2 Functions

nipy.labs.spatial_models.hierarchical_parcellation.hparcel(domain, ldata, nb_parcel, nb_perm=0, niter=5, mu=10.0, dmax=10.0, lamb=100.0, chunk_size=100000.0, verbose=0, initial_mask=None)

Function that performs the parcellation by optimizing the inter-subject similarity while retaining the connectedness within subject and some consistency across subjects.

**Parameters**

- **domain**: discrete_domain.DiscreteDomain instance, :
  yields all the spatial information on the parcelled domain

- **ldata**: list of (n_subj) arrays of shape (domain.size, dim) :
  the feature data used to inform the parcellation

- **nb_parcel**: int, :
  the number of parcels

- **nb_perm**: int, optional, :
  the number of times the parcellation and prfx computation is performed on sign-swaped data

- **niter**: int, optional, :
  number of iterations to obtain the convergence of the method information in the clustering algorithm

- **mu**: float, optional, :
  relative weight of anatomical information

- **dmax**: float optional, :
  some consistency across subjects.
radius of allowed deformations

**lamb**: `float optional`

parameter to control the relative importance of space vs function

**chunksize**: `int, optional`

number of points used in internal sub-sampling

**verbose**: `bool, optional`

verbosity mode

**initial_mask**: `array of shape (domain.size, nb_subj), optional`

initial subject-dependent masking of the domain

**Returns**

Pa: the resulting parcellation structure appended with the labelling:

```
nipy.labs.spatial_models.hierarchical_parcellation.perm_prfx(domain, graphs, features, ldata, initial_mask=None, nb_perm=100, niter=5, dmax=10.0, lamb=100.0, chunksize=100000.0, verbose=1)
```

caveat: assumes that the functional dimension is 1
LABS.SPATIAL_MODELS.HROI

102.1 Module: labs.spatial_models.hroi

Inheritance diagram for nipy.labs.spatial_models.hroi:

```
spatial_models.mroi.SubDomains ───> spatial_models.hroi.HierarchicalROI
```

This module contains the specification of ‘hierarchical ROI’ object, Which is used in spatial models of the library such as structural analysis.

The connection with other classes is not completely satisfactory at the moment: there should be some intermediate classes between ‘Fields’ and ‘hroi’

Author [Bertrand Thirion, 2009-2011] Virgile Fritsch <virgile.fritsch@inria.fr>

102.2 Class

102.3 HierarchicalROI

class nipy.labs.spatial_models.hroi.HierarchicalROI (domain, label, parents, id=None)
Bases: nipy.labs.spatial_models.mroi.SubDomains

Class that handles hierarchical ROIs

**Parameters**

- **k**: int
  Number of ROI in the SubDomains object

- **label**: array of shape (domain.size), dtype=np.int
  An array use to define which voxel belongs to which ROI. The label values greater than -1 correspond to subregions labelling. The labels are recomputed so as to be consecutive integers. The labels should not be accessed outside this class. One has to use the API mapping methods instead.

- **features**: dict {str: list of object, length=self.k}
Describe the voxels features, grouped by ROI

**roi_features**: dict {str: array-like, shape=(self.k, roi_feature_dim)}

Describe the ROI features. A special feature, *id*, is read-only and is used to give an
unique identifier for region, which is persistent through the MROI objects manipulations. On should access the different ROI’s features using ids.

**parents**: np.ndarray, shape(self.k)

self.parents[i] is the index of the parent of the *i*-th ROI.

**TODO**: have the parents as a list of id rather than a list of indices.

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
</table>
| `copy()` | Returns a copy of self.
| `feature_to_voxel_map(fid[, roi, method])` | Convert a feature to a flat voxel-mapping array. |
| `get_coord(id)` | Get coordinates of ROI’s voxels. |
| `get_feature(fid[, id])` | Return a voxel-wise feature, grouped by ROI. |
| `get_id()` | Return ROI’s id list. |
| `get_leaves_id()` | Return the ids of the leaves. |
| `get_local_volume(id)` | Get volume of ROI’s voxels. |
| `get_parents()` | Return the parent of each node in the hierarchy. |
| `get_roi_feature(fid[, id])` | Get ROI size (counted in terms of voxels) |
| `get_size([id, ignore_children])` | Get ROI volume. |
| `get_volume([id, ignore_children])` | Integrate certain feature on each ROI and return the k results. |
| `make_forest()` | Output an nipy forest structure to represent the ROI hierarchy. |
| `make_graph()` | Output an nipy graph structure to represent the ROI hierarchy. |
| `merge_ascending(id_list[, pull_features])` | Remove the non-valid ROIs by including them in |
| `merge_descending([pull_features])` | Remove the items with only one son by including them in their son |
| `plot_feature(fid[, ax])` | Boxplot the distribution of features within ROIs. |
| `recompute_labels()` | Redefine labels so that they are consecutive integers. |
| `reduce_to_leaves()` | Create a new set of rois which are only the leaves of self. |
| `remove_feature(fid)` | Remove a certain feature. |
| `remove_roi_feature(fid)` | Remove a certain ROI feature. |
| `representative_feature(fid[, method, id, ...])` | Compute a ROI representative of a given feature. |
| `select_id(id[, roi])` | Convert a ROI id into an index to be used to index features safely. |
| `select_roi(id_list)` | Returns an instance of HROI with only the subset of chosen ROIs. |
| `set_feature(fid, data[, id, override])` | Append or modify a feature. |
| `set_roi_feature(fid, data[, id, override])` | Append or modify a ROI feature. |
| `to_image([fid, roi, method, descrip])` | Generates a label image that represents self. |

__init__ *(domain, label, parents, id=None)*

Building the HierarchicalROI

**copy ()**

Returns a copy of self.

self.domain is not copied.

**feature_to_voxel_map**( fid=bool, method=’mean’ )

Convert a feature to a flat voxel-mapping array.

Parameters ** fid**: str :
Identifier of the feature to be mapped.

**roi**: bool, optional:

If True, compute the map from a ROI feature.

**method**: str, optional:

Representative feature computation method if *fid* is a feature and *roi* is True.

**Returns**: res: array-like, shape=(domain.size, feature_dim):

A flat array, giving the correspondence between voxels and the feature.

```python
get_coord(id=None)
```

Get coordinates of ROI’s voxels

**Parameters**

*id*: any hashable type:

Id of the ROI from which we want the voxels’ coordinates. Can be None (default) if we want all ROIs’s voxels coordinates.

**Returns**

*coords*: array-like, shape=(roi_size, domain_dimension):

if an id is provided, or list of arrays of shape(roi_size, domain_dimension)

if no id provided (default)

```python
get_feature(fid, id=None)
```

Return a voxel-wise feature, grouped by ROI.

**Parameters**

*fid*: str: Feature to be returned

*id*: any hashable type:

Id of the ROI from which we want to get the feature. Can be None (default) if we want all ROIs’s features.

**Returns**

*feature*: array-like, shape=(roi_size, feature_dim):

if an id is provided, or list of arrays, shape=(roi_size, feature_dim)

if no id provided (default)

```python
get_id()
```

Return ROI’s id list.

Users must access ROIs with the use of the identifiers of this list and the methods that give access to their properties/features.

```python
get_leaves_id()
```

Return the ids of the leaves.

```python
get_local_volume(id=None)
```

Get volume of ROI’s voxels

**Parameters**

*id*: any hashable type:

Id of the ROI from which we want the voxels’ volumes. Can be None (default) if we want all ROIs’s voxels volumes.

**Returns**

*loc_volume*: array-like, shape=(roi_size, ):

if an id is provided, or list of arrays of shape(roi_size,)

if no id provided (default)
get_parents()  
Return the parent of each node in the hierarchy  
The parents are represented by their position in the nodes flat list.  

TODO: The purpose of this class API is not to rely on this order, so we should have self.parents as a list of ids instead of a list of positions

get_roi_feature (fid, id=None)

get_size (id=None, ignore_children=True)
Get ROI size (counted in terms of voxels)

Parameters  
id: any hashable type, optional :
Id of the ROI from which we want to get the size. Can be None (default) if we want all ROIs’s sizes.

ignore_children: bool, optional :
Specify if the size of the node should include (ignore_children = False) or not the one of its children (ignore_children = True).

Returns  
size: int :
if an id is provided, or list of int
if no id provided (default)

get_volume (id=None, ignore_children=True)
Get ROI volume

Parameters  
id: any hashable type, optional :
Id of the ROI from which we want to get the volume. Can be None (default) if we want all ROIs’s volumes.

ignore_children : bool, optional
Specify if the volume of the node should include (ignore_children = False) or not the one of its children (ignore_children = True).

Returns  
volume : float
if an id is provided, or list of float
if no id provided (default)

integrate (fid=None, id=None)
Integrate certain feature on each ROI and return the k results

Parameters  
fid : str
Feature identifier. By default, the 1 function is integrated, yielding ROI volumes.

id: any hashable type :
The ROI on which we want to integrate. Can be None if we want the results for every region.

Returns  
lsum = array of shape (self.k, self.feature[fid].shape[1]), :
The results

make_forest ()
Output an nipy forest structure to represent the ROI hierarchy.
make_graph()

Output an nipy graph structure to represent the ROI hierarchy.

mergeAscending(id_list, pull_features=None)

Remove the non-valid ROIs by including them in their parents when it exists.

Parameters

id_list: list of id (any hashable type):
The id of the ROI to be merged into their parents. Nodes that are their own parent are unmodified.

pull_features: list of str:
List of the ROI features that will be pooled from the children when they are merged into their parents. Otherwise, the receiving parent would keep its own ROI feature.

mergeDescending(pull_features=None)

Remove the items with only one son by including them in their son

Parameters

methods indicates the way possible features are dealt with:

(not implemented yet):

plot_feature(fid, ax=None)

Boxplot the distribution of features within ROIs. Note that this assumes 1-d features.

Parameters

fid: string:
the feature identifier

ax: axis handle, optional:

recompute_labels()

Redefine labels so that they are consecutive integers.

Labels are used as a map to associate voxels to a given ROI. It is an inner object that should not be accessed outside this class. The number of nodes is updated appropriately.

Notes

This method must be called everytime the MROI structure is modified.

reduce_to_leaves()

Create a new set of rois which are only the leaves of self.

Modification of the structure is done in place. One way therefore want to work on a copy a of a given HROI object.

remove_feature(fid)

Remove a certain feature

Parameters

fid: str:
Feature id

Returns

f: object
The removed feature.

remove_roi_feature(fid)

Remove a certain ROI feature.

The id ROI feature cannot be removed.

Returns

f: object
The removed Roi feature.

**representative_feature**(*fid, method='mean', id=None, ignore_children=True, assess_quality=True*)

Compute a ROI representative of a given feature.

**Parameters**

- **fid**: str
  - Feature id

- **method**: str

- **id**: any hashable type
  - Id of the ROI from which we want to extract a representative feature. Can be None (default) if we want to get all ROIs’s representatives.

- **ignore_children**: bool
  - Specify if the volume of the node should include (ignore_children = False) or not the one of its children (ignore_children = True).

- **assess_quality**: bool
  - If True, a new roi feature is created, which represent the quality of the feature representative (the number of non-nan value for the feature over the ROI size). Default is False.

**select_id**(*id, roi=True*)

Convert a ROI id into an index to be used to index features safely.

**Parameters**

- **id**: any hashable type, must be in self.get_id()
  - The id of the region one wants to access.

- **roi**: bool
  - If True (default), return the ROI index in the ROI list. If False, return the indices of the voxels of the ROI with the given id. That way, internal access to self.label can be made.

**Returns**

- **index**: int or np.array of shape (roi.size, )
  - Either the position of the ROI in the ROI list (if roi == True), or the positions of the voxels of the ROI with id id with respect to the self.label array.

**select_roi**(*id_list*)

Returns an instance of HROI with only the subset of chosen ROIs.

The hierarchy is set accordingly.

**Parameters**

- **id_list**: list of id (any hashable type)
  - The id of the ROI to be kept in the structure.

**set_feature**(*fid, data, id=None, override=False*)

Append or modify a feature

**Parameters**

- **fid**: str
  - feature identifier

- **data**: list or array
  - The feature data. Can be a list of self.k arrays of shape(self.size[k], p) or array of shape(self.size[k])
id: any hashable type, optional :

Id of the ROI from which we want to set the feature. Can be None (default) if we want to set all ROIs’s features.

override: bool, optional :

Allow feature overriding

Note that we cannot create a feature having the same name than :

a ROI feature. :

set_roi_feature (fid, data, id=None, override=False)

Append or modify a ROI feature

Parameters fid: str, :

feature identifier

data: list of self.k features or a single feature :

The ROI feature data

id: any hashable type :

Id of the ROI of which we want to set the ROI feature. Can be None (default) if we want to set all ROIs’s ROI features.

override: bool, optional, :

Allow feature overriding

Note that we cannot create a ROI feature having the same name than :

a feature. :

Note that the ‘id‘ feature cannot be modified as an internal :

component. :

to_image (fid=None, roi=False, method='mean', descrip=None)

Generates a label image that represents self.

Parameters fid: str, :

Feature to be represented. If None, a binary image of the MROI domain will be we created.

roi: bool, :

Whether or not to write the desired feature as a ROI one. (i.e. a ROI feature corre-
sponding to fid will be looked upon, and if not found, a representative feature will be computed from the fid feature).

method: str, :

If a feature is written as a ROI feature, this keyword tweaks the way the representative feature is computed.

descrip: str, :

Description of the image, to be written in its header.

Returns nim : nibabel nifti image

Nifti image corresponding to the ROI feature to be written.
Notes

Requires that self.dom is an ddom.NDGridDomain

### 102.4 Functions

**nipy.labs.spatial_models.hroi.HROI_as_discrete_domain_blobs** *(domain, data, threshold=-inf, smin=0, criterion='size')*

Instantiate an HierarchicalROI as the blob decomposition of data in a certain domain.

**Parameters**
- **domain**: discrete_domain.StructuredDomain instance,
  Definition of the spatial context.
- **data**: array of shape (domain.size)
  The corresponding data field.
- **threshold**: float, optional
  Thresholding level.
- **criterion**: string, optional
  To be chosen among ‘size’ or ‘volume’.
- **smin**: float, optional
  A threshold on the criterion.

**Returns**
- **nroi**: HierarchicalROI instance with a ‘signal’ feature.

**nipy.labs.spatial_models.hroi.HROI_from_watershed** *(domain, data, threshold=-inf)*

Instantiate an HierarchicalROI as the watershed of a certain dataset

**Parameters**
- **domain**: discrete_domain.StructuredDomain instance
  Definition of the spatial context.
- **data**: array of shape (domain.size)
  The corresponding data field.
- **threshold**: float, optional
  Thresholding level.

**Returns**
- **nroi**: HierarchicalROI instance
  The HierarchicalROI instance with a seed feature.

**nipy.labs.spatial_models.hroi.hroi_agglomeration** *(input_hroi, criterion='size', smin=0)*

Performs an agglomeration then a selection of regions so that a certain size or volume criterion is satisfied.

**Parameters**
- **input_hroi**: HierarchicalROI instance
  The input hROI
- **criterion**: str, optional
  To be chosen among ‘size’ or ‘volume’
- **smin**: float, optional
The applied criterion

Returns `output_hroi`: HierarchicalROI instance:
```
nipy.labs.spatial_models.hroi.make_hroi_from_subdomain(sub_domain, parents)
```

Instantiate an HROI from a SubDomain instance and parents
103.1 Module: labs.spatial_models.mroi

Inheritance diagram for nipy.labs.spatial_models.mroi:

spatial_models.mroi.SubDomains

103.2 Class

103.3 SubDomains

class nipy.labs.spatial_models.mroi.SubDomains
  
Bases: object

This is a class to represent multiple ROI objects, where the reference to a given domain is explicit.

A multiple ROI object is a set of ROI defined on a given domain, each having its own ‘region-level’ characteristics (ROI features).

Every voxel of the domain can have its own characteristics yet, defined at the ‘voxel-level’, but those features can only be accessed familywise (i.e. the values are grouped by ROI).

Parameters

- **k**: int
  
  Number of ROI in the SubDomains object

- **label**: array of shape (domain.size), dtype=np.int
  
  An array use to define which voxel belongs to which ROI. The label values greater than -1 correspond to subregions labelling. The labels are recomputed so as to be consecutive integers. The labels should not be accessed outside this class. One has to use the API mapping methods instead.

- **features**: dict {str: list of object, length=self.k}
Describe the voxels features, grouped by ROI

**roi_features** : dict {str: array-like, shape=(self.k, roi_feature_dim)}

Describe the ROI features. A special feature, *id*, is read-only and is used to give an unique identifier for region, which is persistent through the MROI objects manipulations. On should access the different ROI’s features using *ids*.

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>copy()</td>
<td>Returns a copy of self.</td>
</tr>
<tr>
<td>feature_to_voxel_map(fid[, roi, method])</td>
<td>Convert a feature to a flat voxel-mapping array.</td>
</tr>
<tr>
<td>get_coord([id])</td>
<td>Get coordinates of ROI’s voxels</td>
</tr>
<tr>
<td>get_feature(fid[, id])</td>
<td>Return a voxel-wise feature, grouped by ROI.</td>
</tr>
<tr>
<td>get_id()</td>
<td>Return ROI’s id list.</td>
</tr>
<tr>
<td>get_local_volume([id])</td>
<td>Get volume of ROI’s voxels</td>
</tr>
<tr>
<td>get_roi_feature(fid[, id])</td>
<td></td>
</tr>
<tr>
<td>get_size([id])</td>
<td>Get ROI size (counted in terms of voxels)</td>
</tr>
<tr>
<td>get_volume([id])</td>
<td>Get ROI volume</td>
</tr>
<tr>
<td>integrate([fid, id])</td>
<td>Integrate certain feature on each ROI and return the k results</td>
</tr>
<tr>
<td>plot_feature(fid[, ax])</td>
<td>Boxplot the distribution of features within ROIs.</td>
</tr>
<tr>
<td>recompute_labels()</td>
<td>Redefine labels so that they are consecutive integers.</td>
</tr>
<tr>
<td>remove_feature(fid)</td>
<td>Remove a certain feature</td>
</tr>
<tr>
<td>remove_roi_feature(fid)</td>
<td>Remove a certain ROI feature</td>
</tr>
<tr>
<td>representative_feature_feature(fid[, method, id, ...])</td>
<td>Compute a ROI representative of a given feature.</td>
</tr>
<tr>
<td>select_id(id[, roi])</td>
<td>Convert a ROI id into an index to be used to index features safely.</td>
</tr>
<tr>
<td>select_roi(id_list)</td>
<td>Returns an instance of MROI with only the subset of chosen ROIs.</td>
</tr>
<tr>
<td>set_feature(fid, data[, id, override])</td>
<td>Append or modify a feature</td>
</tr>
<tr>
<td>set_roi_feature(fid, data[, id, override])</td>
<td>Append or modify a ROI feature</td>
</tr>
<tr>
<td>to_image([fid, roi, method, descrip])</td>
<td>Generates a label image that represents self.</td>
</tr>
</tbody>
</table>

```
__init__ (domain, label, id=None)
Initialize subdomains instance
```

**Parameters**

- **domain:** ROI instance:
  - defines the spatial context of the SubDomains
  - label: array of shape (domain.size), dtype=np.int,
    - An array use to define which voxel belongs to which ROI. The label values greater than - 1 correspond to subregions labelling. The labels are recomputed so as to be consecutive integers. The labels should not be accessed outside this class. One has to use the select_id() mapping method instead.
  - id: array of shape (n_roi):
    - Define the ROI identifiers. Once an id has been associated to a ROI it becomes impossible to change it using the API. Hence, one should access ROI through their id to avoid hazardous manipulations.

```
copy ()
Returns a copy of self.
```

Note that self.domain is not copied.
**feature_to_voxel_map** *(fid, roi=False, method='mean')*

Convert a feature to a flat voxel-mapping array.

**Parameters**

- **fid**: str
  
  Identifier of the feature to be mapped.

- **roi**: bool, optional
  
  If True, compute the map from a ROI feature.

- **method**: str, optional
  
  Representative feature computation method if *fid* is a feature and *roi* is True.

**Returns**

- **res**: array-like, shape=(domain.size, feature_dim)
  
  A flat array, giving the correspondence between voxels and the feature.

**get_coord** *(id=None)*

Get coordinates of ROI’s voxels

**Parameters**

- **id**: any hashable type
  
  Id of the ROI from which we want the voxels’ coordinates. Can be None (default) if we want all ROIs’s voxels coordinates.

**Returns**

- **coords**: array-like, shape=(roi_size, domain_dimension)
  
  if an id is provided, or list of arrays of shape(roi_size, domain_dimension)

  if no id provided (default)

**get_feature** *(fid, id=None)*

Return a voxel-wise feature, grouped by ROI.

**Parameters**

- **fid**: str, :
  
  Feature to be returned

- **id**: any hashable type
  
  Id of the ROI from which we want to get the feature. Can be None (default) if we want all ROIs’s features.

**Returns**

- **feature**: array-like, shape=(roi_size, feature_dim)
  
  if an id is provided, or list of arrays, shape=(roi_size, feature_dim)

  if no id provided (default)

**get_id** *

Return ROI’s id list.

Users must access ROIs with the use of the identifiers of this list and the methods that give access to their properties/features.

**get_local_volume** *(id=None)*

Get volume of ROI’s voxels

**Parameters**

- **id**: any hashable type
  
  Id of the ROI from which we want the voxels’ volumes. Can be None (default) if we want all ROIs’s voxels volumes.

**Returns**

- **loc_volume**: array-like, shape=(roi_size, ),
  
  if an id is provided, or list of arrays, shape=(roi_size, feature_dim)

  if no id provided (default)
if an id is provided, or list of arrays of shape(roi_size, )
if no id provided (default)

get_roi_feature (fid, id=None)
get_size (id=None)
Get ROI size (counted in terms of voxels)

Parameters  id: any hashable type :
           Id of the ROI from which we want to get the size. Can be None (default) if we want all
           ROIs’s sizes.

Returns  size: int :
            if an id is provided, or list of int
            if no id provided (default)

get_volume (id=None)
Get ROI volume

Parameters  id: any hashable type :
           Id of the ROI from which we want to get the volume. Can be None (default) if we want all
           ROIs’s volumes.

Returns  volume : float
            if an id is provided, or list of float
            if no id provided (default)

integrate (fid=None, id=None)
Integrate certain feature on each ROI and return the k results

Parameters  fid : str
            Feature identifier. By default, the 1 function is integrated, yielding ROI volumes.

            id: any hashable type :
            The ROI on which we want to integrate. Can be None if we want the results for every
            region.

Returns  lsum = array of shape (self.k, self.feature[fid].shape[1]), :
            The results

plot_feature (fid, ax=None)
Boxplot the distribution of features within ROIs. Note that this assumes 1-d features.

Parameters  fid: string :
            the feature identifier

            ax: axis handle, optional :

recompute_labels ()
Redefine labels so that they are consecutive integers.

Labels are used as a map to associate voxels to a given ROI. It is an inner object that should not be accessed
outside this class. The number of nodes is updated appropriately.
Notes

This method must be called every time the MROI structure is modified.

```python
remove_feature(fid)
```

Remove a certain feature

**Parameters**

- **fid**: str
  - Feature id

**Returns**

- **f**: object
  - The removed feature.

```python
remove_roi_feature(fid)
```

Remove a certain ROI feature.

The id ROI feature cannot be removed.

**Returns**

- **f**: object
  - The removed Roi feature.

```python
representative_feature(fid, method='mean', id=None, assess_quality=False)
```

Compute a ROI representative of a given feature.

**Parameters**

- **fid**: str
  - Feature id
- **method**: str, optional
- **id**: any hashable type, optional
  - Id of the ROI from which we want to extract a representative feature. Can be None (default) if we want to get all ROIs’s representatives.
- **assess_quality**: bool, optional
  - If True, a new roi feature is created, which represent the quality of the feature representative (the number of non-nan value for the feature over the ROI size). Default is False.

**Returns**

- **summary_feature**: np.ndarray, shape=(self.k, feature_dim)
  - Representative feature computed according to `method`.

```python
select_id(id, roi=True)
```

Convert a ROI id into an index to be used to index features safely.

**Parameters**

- **id**: any hashable type, must be in self.get_id()
  - The id of the region one wants to access.
- **roi**: bool
  - If True (default), return the ROI index in the ROI list. If False, return the indices of the voxels of the ROI with the given id. That way, internal access to self.label can be made.

**Returns**

- **index**: int or np.array of shape (roi.size,)
  - Either the position of the ROI in the ROI list (if roi == True), or the positions of the voxels of the ROI with id `id` with respect to the self.label array.
select_roi \((id\_list)\)

Returns an instance of MROI with only the subset of chosen ROIs.

**Parameters**

- **id\_list**: list of id (any hashable type)
  
  The id of the ROI to be kept in the structure.

set_feature \((fid, data, id=None, override=False)\)

Append or modify a feature

**Parameters**

- **fid**: str
  
  feature identifier

- **data**: list or array
  
  The feature data. Can be a list of self.k arrays of shape(self.size[k], p) or array of shape(self.size[k])

- **id**: any hashable type, optional
  
  Id of the ROI from which we want to set the feature. Can be None (default) if we want to set all ROIs’s features.

- **override**: bool, optional
  
  Allow feature overriding

  **Note that we cannot create a feature having the same name than a ROI feature.**

set_roi_feature \((fid, data, id=None, override=False)\)

Append or modify a ROI feature

**Parameters**

- **fid**: str
  
  feature identifier

- **data**: list of self.k features or a single feature
  
  The ROI feature data

- **id**: any hashable type
  
  Id of the ROI of which we want to set the ROI feature. Can be None (default) if we want to set all ROIs’s ROI features.

- **override**: bool, optional
  
  Allow feature overriding

  **Note that we cannot create a ROI feature having the same name than a feature.**

  **Note that the ‘id’ feature cannot be modified as an internal component.**

to_image \((fid=None, roi=False, method='mean', descrip=None)\)

Generates a label image that represents self.

**Parameters**

- **fid**: str
  
  Feature to be represented. If None, a binary image of the MROI domain will be created.

- **roi**: bool
  
  Indicates whether the image should be of type ROI or MROI.
Whether or not to write the desired feature as a ROI one. (i.e. a ROI feature corresponding to fid will be looked upon, and if not found, a representative feature will be computed from the fid feature).

**method:** str,

If a feature is written as a ROI feature, this keyword tweaks the way the representative feature is computed.

**descrip:** str,

Description of the image, to be written in its header.

**Returns** nim : nibabel nifti image

Nifti image corresponding to the ROI feature to be written.

**Notes**

Requires that self.dom is an ddom.NDGridDomain

### 103.4 Functions

**nipy.labs.spatial_models.mroi.subdomain_from_array** (labels, affine=None, nn=0)

Return a SubDomain from an n-d int array

**Parameters**

**label:** np.array instance :

A supposedly boolean array that yields the regions.

**affine:** np.array, optional :

Affine transform that maps the array coordinates to some embedding space by default, this is np.eye(dim+1, dim+1).

**nn:** int, :

Neighboring system considered. Unused at the moment.

**Notes**

Only labels > -1 are considered.

**nipy.labs.spatial_models.mroi.subdomain_from_balls** (domain, positions, radii)

Create discrete ROIs as a set of balls within a certain coordinate systems.

**Parameters**

**domain:** StructuredDomain instance, :

the description of a discrete domain

**positions:** array of shape(k, dim): :

the positions of the balls

**radii:** array of shape(k): :

the sphere radii

**nipy.labs.spatial_models.mroi.subdomain_from_image** (mim, nn=18)

Return a SubDomain instance from the input mask image.
Parameters

- `mim`: NiftiImage instance, or string path toward such an image:
  - supposedly a label image

- `nn`: int, optional:
  - Neighboring system considered from the image can be 6, 18 or 26.

Returns

The MultipleROI instance:

Notes

Only labels > -1 are considered

`nipy.labs.spatial_models.mroi.subdomain_from_position_and_image(nim, pos)`

Keep the set of labels of the image corresponding to a certain index so that their position is closest to the prescribed one.

Parameters

- `mim`: NiftiImage instance, or string path toward such an image:
  - supposedly a label image

- `pos`: array of shape(3) or list of length 3,:
  - the prescribed position
104.1 Module: labs.spatial_models.parcel_io

Utility functions for multi-subject parcellation: this basically uses nipy io lib to perform IO operation in parcel definition processes

104.2 Functions

nipy.labs.spatial_models.parcel_io.fixed_parcellation(mask_image, betas, nbparcel, nn=6, method='ward', write_dir=None, mu=10.0, verbose=0, fullpath=None)

Fixed parcellation of a given dataset

Parameters domain/mask_image :

betas: list of paths to activation images from the subject :

nbparcel, int : number of desired parcels

nn=6: number of nearest neighbors to define the image topology :

(6, 18 or 26)

method='ward': clustering method used, to be chosen among :

‘ward’, ‘gkm’, ‘ward_and_gkm’ ‘ward’: Ward’s clustering algorithm ‘gkm’: Geodesic k-means algorithm, random initialization ‘gkm_and_ward’: idem, initialized by Ward’s clustering

write_dir: string, topional, write directory. :

If fullpath is None too, then no file output.

mu = 10., float: the relative weight of anatomical information :

verbose=0: verbosity mode :

fullpath= None, string, :

path of the output image If write_dir and fullpath are None then no file output. If only fullpath is None then it is the write dir + a name depending on the method.
Notes

Ward’s method takes time (about 6 minutes for a 60K voxels dataset)
Geodesic k-means is ‘quick and dirty’
Ward’s + GKM is expensive but quite good
To reduce CPU time, rather use nn=6 (especially with Ward)
nipy.labs.spatial_models.parcel_io.mask_parcellation(mask_images, nb_parcel,
threshold=0, output_image=None)

Performs the parcellation of a certain mask

Parameters

- mask_images: string or Nifti1Image or list of strings/Nifti1Images, :
  paths of mask image(s) that define(s) the common space.
- nb_parcel: int, :
  number of desired parcels
- threshold: float, optional, :
  level of intersection of the masks
- output_image: string, optional :
  path of the output image

Returns

- wim: Nifti1Imagine instance, representing the resulting parcellation :

nipy.labs.spatial_models.parcel_io.parcel_input(mask_images, learning_images,
ths=0.5, fdim=None)

Instantiating a Parcel structure from a given set of input

Parameters

- mask_images: string or Nifti1Image or list of strings/Nifti1Images, :
  paths of mask image(s) that define(s) the common space.
- learning_images: (nb_subject-) list of (nb_feature-) list of strings, :
  paths of feature images used as input to the parcellation procedure
- ths=0.5: threshold to select the regions that are common across subjects, :
  if ths = .5, the threshold is half the number of subjects
- fdim: int, optional :
  if nb_feature (the dimension of the data) used in subsequent analyses if greater than fdim, a PCA is performed to reduce the information in the data By default, no reduction is performed

Returns

- domain: discrete_domain.DiscreteDomain instance
  that stores the spatial information on the parcelled domain
- feature: (nb_subect-) list of arrays of shape (domain.size, fdim) :
  feature information available to parcellate the data
nipy.labs.spatial_models.parcel_io.parcellation_based_analysis(Pa, test_images,
test_id='one_sample',
rfx_path=None,
condition_id='',
swd=None)

This function computes parcel averages and RFX at the parcel-level

**Parameters**

- **Pa**: MultiSubjectParcellation instance
  - the description of the parcellation
- **test_images**: (Pa.nb_subj-) list of paths
  - paths of images used in the inference procedure
- **test_id**: string, optional
  - if test_id='one_sample', the one_sample statistic is computed otherwise, the parcel-based signal averages are returned
- **rfx_path**: string optional
  - path of the resulting one-sample test image, if applicable
- **swd**: string, optional
  - output directory used to compute output path if rfx_path is not given
- **condition_id**: string, optional
  - contrast/condition id used to compute output path

**Returns**

- **test_data**: array of shape(Pa.nb_parcel, Pa.nb_subj)
  - the parcel-level signal average if test is not ‘one_sample’
- **prfx**: array of shape(Pa.nb_parcel),
  - the one-sample t-value if test_id is ‘one_sample’

nipy.labs.spatial_models.parcel_io.write_parcellation_images(Pa, 
template_path=None,
indiv_path=None,
subject_id=None,
swd=None)

Write images that describe the spatial structure of the parcellation

**Parameters**

- **Pa**: MultiSubjectParcellation instance,
  - the description of the parcellation
- **template_path**: string, optional
  - path of the group-level parcellation image
- **indiv_path**: list of strings, optional
  - paths of the individual parcellation images
- **subject_id**: list of strings of length Pa.nb_subj
  - subject identifiers, used to infer the paths when not available
- **swd**: string, optional
  - output directory used to infer the paths when these are not available
105.1 Module: labs.spatial_models.parcellation

Inheritance diagram for nipy.labs.spatial_models.parcellation:

```
spatial_models.parcellation.MultiSubjectParcellation
```

Generic Parcellation class: Contains all the items that define a multi-subject parcellation

Author: Bertrand Thirion, 2005-2008

TODO: add a method 'global field', i.e. non-subject-specific info

105.2 MultiSubjectParcellation

```python
class nipy.labs.spatial_models.parcellation.MultiSubjectParcellation(
domain,
template_labels=None,
individual_labels=None,
nb_parcel=None
)
```

Bases: object

MultiSubjectParcellation class are used to represent parcels that can have different spatial different contours in a given group of subject It consists of self.domain: the specification of a domain self.template_labels the specification of a template parcellation self.individual_labels the specification of individual parcellations

fixme: should inherit from mroi.MultiROI

Methods

```
check() Performs an elementary check on self
```

Continued on next page
Table 105.1 – continued from previous page

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>copy()</code></td>
<td>Returns a copy of self</td>
</tr>
<tr>
<td><code>get_feature(fid)</code></td>
<td>Get feature defined by <code>fid</code></td>
</tr>
<tr>
<td><code>make_feature(fid, data)</code></td>
<td>Compute parcel-level averages of data</td>
</tr>
<tr>
<td><code>population()</code></td>
<td>Returns the counting of labels per voxel per subject</td>
</tr>
<tr>
<td><code>set_feature(fid, data)</code></td>
<td>Set feature defined by <code>fid</code> and <code>data</code> into <code>self</code></td>
</tr>
<tr>
<td><code>set_individual_labels(individual_labels)</code></td>
<td>Set individual labels <code>individual_labels</code> into <code>self</code></td>
</tr>
<tr>
<td><code>set_template_labels(template_labels)</code></td>
<td>Set template labels <code>template_labels</code> into <code>self</code></td>
</tr>
</tbody>
</table>

__init__ (domain, template_labels=None, individual_labels=None, nb_parcel=None)

Initialize multi-subject parcellation

Parameters:
- `domain`: discrete_domain.DiscreteDomain instance,
  definition of the space considered in the parcellation
- `template_labels`: array of shape `domain.size`, optional,
  definition of the template labelling
- `individual_labels`: array of shape `(domain.size, nb_subjects)`, optional,
  the individual parcellations corresponding to the template
- `nb_parcel`: int, optional,
  number of parcels in the model can be inferred as `template_labels.max()+1`, or 1 by default cannot be smaller than `template_labels.max()+1`

`check()`
Perform elementary check on `self`

`copy()`
Returns a copy of `self`

`get_feature(fid)`
Get feature defined by `fid`

Parameters:
- `fid`: string, the feature identifier

`make_feature(fid, data)`
Compute parcel-level averages of data

Parameters:
- `fid`: string, the feature identifier
- `data`: array of shape `(self.domain.size, self.nb_subj, dim)` or:
  `(self.domain.size, self.nb_subj)` Some information at the voxel level

Returns:
- `pfeature`: array of shape `(self.nb_parcel, self.nbsubj, dim)`,
  the computed feature data

`population()`
Returns the counting of labels per voxel per subject

Returns:
- `population`: array of shape `(self.nb_parcel, self.nb_subj)`

`set_feature(fid, data)`
Set feature defined by `fid` and `data` into `self`

Parameters:
- `fid`: string,
  the feature identifier
data: array of shape (self.nbParcel, self.nbSubj, dim) or:

(self.nbParcel, self.nbSubj)

the data to be set as parcel- and subject-level information

set_individual_labels(individual_labels)

set_template_labels(template_labels)
LABS.SPATIAL_MODELS.STRUCTURAL_BFLS

106.1 Module: labs.spatial_models.structural_bfls

Inheritance diagram for nipy.labs.spatial_models.structural_bfls:

The main routine of this module implement the LandmarkRegions class, that is used to represent Regions of interest at the population level (in a template space).

This has been used in Thirion et al. Structural Analysis of fMRI Data Revisited: Improving the Sensitivity and Reliability of fMRI Group Studies. IEEE TMI 2007

Author : Bertrand Thirion, 2006-2010

106.2 LandmarkRegions

class nipy.labs.spatial_models.structural_bfls.LandmarkRegions (domain, k, indiv_coord, subj, id='')

Bases: object

This class is intended to represent a set of inter-subject regions It should inherit from some abstract multiple ROI class, not implemented yet.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>centers()</td>
<td>returns the average of the coordinates for each region</td>
</tr>
<tr>
<td>density(k[, coord, dmax, dof])</td>
<td>Posterior density of component k</td>
</tr>
<tr>
<td>get_feature(fid)</td>
<td></td>
</tr>
<tr>
<td>homogeneity()</td>
<td>returns the mean distance between points within each LR</td>
</tr>
</tbody>
</table>

Continued on next page
Table 106.1 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>hpd(k[, coord, pval, dmax])</code></td>
<td>Sample the posterior probability of being in k</td>
</tr>
<tr>
<td><code>map_label([coord, pval, dmax])</code></td>
<td>Sample the set of landmark regions</td>
</tr>
<tr>
<td><code>prevalence_density()</code></td>
<td>Returns a weighted map of self.prevalence</td>
</tr>
<tr>
<td><code>roi_confidence([ths, fid])</code></td>
<td>assuming that a certain feature fid field has been set</td>
</tr>
<tr>
<td><code>roi_prevalence([fid])</code></td>
<td>assuming that fid='confidence' field has been set</td>
</tr>
<tr>
<td><code>set_feature(fid, data)</code></td>
<td></td>
</tr>
<tr>
<td><code>show()</code></td>
<td>function to print basic information on self</td>
</tr>
<tr>
<td><code>weighted_feature_density(feature)</code></td>
<td>Given a set of feature values, produce a weighted feature map,</td>
</tr>
</tbody>
</table>

```python
__init__(domain, k, indiv_coord, subj, id='')
```

Building the landmark region

**Parameters**
- **domain**: ROI instance:
  - defines the spatial context of the SubDomains
- **k**: int, the number of regions considered:
- **indiv_coord**: k-length list of arrays, optional:
  - coordinates of the nodes in some embedding space.
- **subj**: k-length list of integers:
  - these correspond to and ROI feature: the subject index of individual regions
- **id**: string, optional, identifier:

```python
centers()
```

returns the average of the coordinates for each region

```python
density(k, coord=None, dmax=1.0, dof=10)
```

Posterior density of component k

**Parameters**
- **k**: int, less or equal to self.k:
  - reference component
- **coord**: array of shape(n, self.dom.em_dim), optional:
  - a set of input coordinates
- **dmax**: float, optional:
  - regularizaing constant for the variance estimation
- **dof**: float, optional:
  - strength of the regularization

**Returns**
- **pd**: array of shape(n):
  - the posterior density that has been computed
- **delta**: array of shape(n):
  - the quadratic term in the gaussian model

```python
get_feature(fid)
```

```python
homogeneity()
```

returns the mean distance between points within each LR

```python
hpd(k, coord=None, pval=0.95, dmax=1.0)
```

Sample the posterior probability of being in k on a grid defined by cs, assuming that the roi is an ellipsoid
Parameters  

- **k**: int, less or equal to self.k : reference component

- **coord**: array of shape(n,dim), optional : a set of input coordinates

- **pval**: float<1, optional, :
  - cutoff for the CR

- **dmax=1.0**: an upper bound for the spatial variance : to avoid degenerate variance

Returns  

- **hpd array of shape(n) that yields the value** :

**map_label (coord=None, pval=0.95, dmax=1.0)**

Sample the set of landmark regions on the proposed coordinate set cs, assuming a Gaussian shape

Parameters  

- **coord**: array of shape(n,dim), optional, : a set of input coordinates

- **pval**: float in [0,1), optional :
  - cutoff for the CR, i.e. highest posterior density threshold

- **dmax**: an upper bound for the spatial variance :
  - to avoid degenerate variance

Returns  

- **label**: array of shape (n): the posterior labelling :

**prevalence_density ()**

Returns a weighted map of self.prevalence

Returns  

- **wp**: array of shape(n_samples) :

**roi_confidence (ths=0, fid='confidence')**

assuming that a certain feature fid field has been set as a discrete feature, this creates an approximate p-value that states how confident one might that the LR is defined in at least ths individuals if confidence is not defined as a discrete_feature, it is assumed to be 1.

Parameters  

- **ths**: integer that yields the representativity threshold :

Returns  

- **pvals**: array of shape self.k :
  - the p-values corresponding to the ROIs

**roi_prevalence (fid='confidence')**

assuming that fid='confidence' field has been set as a discrete feature, this creates the expectancy of the confidence measure i.e. expected numberof detection of the roi in the observed group

Returns  

- **confid**: array of shape self.k :
  - the population_prevalence

**set_feature (fid, data)**

**show ()**

function to print basic information on self

**weighted_feature_density (feature)**

Given a set of feature values, produce a weighted feature map, where roi-level features are mapped smoothly based on the density of the components
Parameters  feature: array of shape (self.k), :
  the information to map

Returns  wsm: array of shape(self.shape) :

nipy.labs.spatial_models.structural_bfls.build_LR(bf, thq=0.95, ths=0, dmax=1.0, verbose=0)

Given a list of hierarchical ROIs, and an associated labelling, this creates an Amer structure which groups ROIs with the same label.

Parameters  bf : list of nipy.labs.spatial_models.hroi.Nroi instances
  it is assumed that each list corresponds to one subject each HierarchicalROI is assumed to have the roi_features 'position', 'label' and 'posterior_proba' defined

thq=0.95, ths=0 defines the condition (c): :
  (c) A label should be present in ths subjects with a probability>thq in order to be valid

dmax: float optional, :
  regularizing constant that defines a prior on the region extent

Returns  LR : None or structural_bfls.LR instance
  describing a cross-subject set of ROIs. If inference yields a null result, LR is set to None

newlabel: a relabelling of the individual ROIs, similar to u, :
  which discards labels that do not fulfill the condition (c)
107.1 Module: labs.statistical_mapping

Inheritance diagram for nipy.labs.statistical_mapping:

```python
labs.statistical_mapping.LinearModel
```

107.2 Class

107.3 LinearModel

class nipy.labs.statistical_mapping.LinearModel(data, design_matrix, mask=None, formula=None, model='spherical', method=None, niter=2)

Bases: object

Methods

```python
contrast(vector)  Compute images of contrast and contrast variance.
dump(filename)   Dump GLM fit as npz file.
__init__(data, design_matrix, mask=None, formula=None, model='spherical', method=None, niter=2)
contrast(vector)  Compute images of contrast and contrast variance.
def_model = 'spherical'
def_niter = 2
```
dump (filename)
    Dump GLM fit as npz file.

107.4 Functions

nipy.labs.statistical_mapping.bonferroni(p, n)

nipy.labs.statistical_mapping.cluster_stats(zimg, mask, height_th, height_control='fpr',
cluster_th=0, nulls={})
    Return a list of clusters, each cluster being represented by a dictionary. Clusters are sorted by descending size order. Within each cluster, local maxima are sorted by descending depth order.

    Parameters
        zimg: z-score image :
            mask: mask image :
        height_th: cluster forming threshold :
        height_control: string :
            false positive control meaning of cluster forming threshold:
                'fpr'|'fdr'|'bonferroni'|'none'
        cluster_th: cluster size threshold :
        null_s : cluster-level calibration method: None'rft'array

Notes

This works only with three dimensional data

nipy.labs.statistical_mapping.get_3d_peaks(image, mask=None, threshold=0.0, nn=18, order_th=0)
    returns all the peaks of image that are with the mask and above the provided threshold

    Parameters
        image, (3d) test image :
            mask=None, (3d) mask image :
                By default no masking is performed
            threshold=0.0, float, threshold value above which peaks are considered :
            nn=18, int, number of neighbours of the topological spatial model :
            order_th=0, int, threshold on topological order to validate the peaks :

Returns
    peaks, a list of dictionaries, where each dict has the fields:
        vals, map value at the peak :
        order, topological order of the peak :
        ijk, array of shape (1,3) grid coordinate of the peak :
        pos, array of shape (n_maxima,3) mm coordinates (mapped by affine) :
    of the peaks

nipy.labs.statistical_mapping.linear_model_fit(data_images, mask_images, design_matrix, vector)
    Helper function for group data analysis using arbitrary design matrix
nipy.labs.statistical_mapping.onesample_test(data_images, vardata_images, mask_images, stat_id, permutations=0, cluster_forming_th=0.01)

Helper function for permutation-based mass univariate onesample group analysis.

nipy.labs.statistical_mapping.prepare_arrays(data_images, vardata_images, mask_images)

nipy.labs.statistical_mapping.simulated_pvalue(t, simu_t)

nipy.labs.statistical_mapping.twosample_test(data_images, vardata_images, mask_images, labels, stat_id, permutations=0, cluster_forming_th=0.01)

Helper function for permutation-based mass univariate twosample group analysis. Labels is a binary vector (1-2). Regions more active for group 1 than group 2 are inferred.
LABS.UTILS.RANDOM_THRESHOLD

108.1 Module: labs.utils.random_threshold

108.2 Functions

nipy.labs.utils.random_threshold.isolated(XYZ, k=18)
Outputs an index I of isolated points from their integer coordinates, XYZ (3, n), and under k-connectivity, k = 6, 18 or 26.

nipy.labs.utils.random_threshold.randthresh(Y, K, p=inf, stop=False, verbose=False, varwind=False, knownull=True)
Wrapper for random threshold functions (without connexity constraints)

Parameters
  Y: array of shape (n,) Observations :
    K: int:
      Some positive integer (lower bound on the number of null hypotheses)
    p: float, lp norm :
    stop <bool> Stop when minimum is attained (save computation time) :
    verbose <bool> ‘Chatty’ mode :
    varwind <bool> Varying window variant (vs. fixed window, with width K) :
    knownull <bool> :
      Known null distribution (observations assumed Exp(1) under H0) versus unknown (observations assumed Gaussian under H0)

Returns:

A dictionary D containing the following fields:

“C” (n-K):
  Lp norm of partial sums fluctuation about their conditional expectation

“thresh” <float> Detection threshold :

“detect” (k,) Index of detected activations :

“v” <float> Estimated null variance (if knownull is False) :
Notes

Random thresholding is performed only if null hypothesis of no activations is rejected at level 5%

```
nipy.labs.utils.random_threshold.randthresh_connex(Y, K, XYZ, p=inf, stop=False, verbose=False, varwind=False, knownull=True)
```

Wrapper for random threshold functions under connexity constraints

**Parameters**

- `Y (n,) Observations`
- `K <int>`: Some positive integer (lower bound on the number of null hypotheses)
- `XYZ (3,n) voxel coordinates`
- `p <float> lp norm`
- `stop <bool> Stop when minimum is attained (save computation time)`
- `verbose <bool> ‘Chatty’ mode`
- `varwind <bool> Varying window variant (vs. fixed window, with width K)`
- `knownull <bool>`: Known null distribution (observations assumed Exp(1) under H0) versus unknown (observations assumed Gaussian under H0)

**Returns**

A dictionary `D` containing the following fields:

- “C” (n-K): Lp norm of partial sums fluctuation about their conditional expectation
- “thresh” <float> Detection threshold
- “detect” (ncoeffs,) Index of detected voxels

Notes

Random thresholding is performed only if null hypothesis of no activations is rejected at level 5%

```
nipy.labs.utils.random_threshold.randthresh_fixwind_gaussnull(Y, K, p=inf, stop=False, one_sided=False, verbose=False)
```

Random threshold with fixed window and null gaussian distribution

**Parameters**

- `Y array of shape (n,)`: Observations (assumed Gaussian under H0, with unknown variance)
- `K, int, Some positive integer`: (lower bound on the number of null hypotheses)
- `p, float, lp norm`
- `stop: bool,`: Stop when minimum is attained (save computation time)
- `one_sided: bool,`:
If nonzero means are positive only (vs. positive or negative)

Returns  C array of shape (n-K):

Lp norm of partial sums fluctuation about their conditional expectation

nipy.labs.utils.random_threshold.randthresh_fixwind_gaussnull_connex(X, K, XYZ, p=inf, stop=False, verbose=False)

Random threshold with fixed-window and gaussian null distribution, using connexity constraint on non-null set.

Parameters  X (n,): Observations (assumed Gaussian under H0):

XYZ (3,n): voxel coordinates:

K <int>:

Some positive integer (lower bound on the number of null hypotheses)

p <float>: Lp norm:

stop <bool>: Stop when minimum is attained (save computation time):

Returns  C (n-K):

Lp norm of partial sums fluctuation about their conditional expectation

nipy.labs.utils.random_threshold.randthresh_fixwind_gaussnull_connex(X, K, XYZ, p=inf, stop=False, verbose=False)

Random threshold with fixed-window and gaussian null distribution, using connexity constraint on non-null set.

Parameters  X (n,): Observations (assumed Gaussian under H0):

XYZ (3,n): voxel coordinates:

K <int>:

Some positive integer (lower bound on the number of null hypotheses)

p <float>: Lp norm:

stop <bool>: Stop when minimum is attained (save computation time):

Returns  C (n-K):

Lp norm of partial sums fluctuation about their conditional expectation

nipy.labs.utils.random_threshold.randthresh_fixwind_gaussnull_connex(X, K, XYZ, p=inf, stop=False, verbose=False)

Random threshold with fixed-window and known null distribution

Parameters  X (n,): Observations (must be Exp(1) under H0):

K <int>:

Some positive integer (lower bound on the number of null hypotheses)

p <float>: Lp norm:

stop <bool>: Stop when minimum is attained (save computation time):

Returns  C (n-K):

Lp norm of partial sums fluctuation about their conditional expectation

nipy.labs.utils.random_threshold.randthresh_fixwind_knownull_connex(X, K, p=inf, stop=False, verbose=False)

Random threshold with fixed-window and known null distribution, using connexity constraint on non-null set.

Parameters  X (n,): Observations (must be Exp(1) under H0):

XYZ (3,n): voxel coordinates:

K <int>:

Some positive integer (lower bound on the number of null hypotheses)

p <float>: Lp norm:

stop <bool>: Stop when minimum is attained (save computation time):

Returns  C (n-K):

Lp norm of partial sums fluctuation about their conditional expectation

nipy.labs.utils.random_threshold.randthresh_fixwind_knownull_connex(X, K, XYZ, p=inf, stop=False, verbose=False)
Returns \( C (n-K) \): 

Lp norm of partial sums fluctuation about their conditional expectation

```
nipy.labs.utils.random_threshold.randthresh_main(Y, K, XYZ=None, p=inf, varwind=False, knownull=True, stop=False, verbose=False)
```

Wrapper for random threshold functions

**Parameters**

\( Y \): array of shape \((n,)\), Observations :

\( K \): int, Some positive integer :

(lower bound on the number of null hypotheses)

\( XYZ \): array of shape \((3, n)\) voxel coordinates. :

If not empty, connexity constraints are used on the non-null set

\( p \): float, optional, lp norm :

\( \text{varwind} \): bool, :

Varying window variant (vs. fixed window, with width \( K \))

\( \text{knownull} \): bool, optional, :

Known null distribution (observations assumed \( \text{Exp}(1) \) under \( H_0 \)) versus unknown (observations assumed Gaussian under \( H_0 \))

\( \text{stop} \): bool, optional :

Stop when minimum is attained (save computation time)

\( \text{verbose} \): bool, ‘Chatty’ mode :

**Returns** \( D \): dict

containing the following fields: * “C” \((n-K)\) array Lp norm of partial sums fluctuation about their conditional expectation

- “thresh” \(<\text{float}>\) Detection threshold
- “detect” \((k,)\) Index of detected activations

**Notes**

Random thresholding is performed only if null hypothesis of no activations is rejected at level 5%

```
nipy.labs.utils.random_threshold.randthresh_varwind_gaussnull(Y, K, p=inf, stop=False, one_sided=False, verbose=False)
```

Random threshold with fixed window and gaussian null distribution

**Parameters**

\( Y \ (n,) \) Observations (assumed Gaussian under \( H_0 \), with unknown variance) :

\( K \ <\text{int}> \) :

Some positive integer (lower bound on the number of null hypotheses)

\( p \ <\text{float}> \ lp \ norm \) :

\( \text{stop} \ <\text{bool}> \) Stop when minimum is attained (save computation time) :
one_sided <bool>:
If nonzero means are positive only (vs. positive or negative)

Returns C (n-K):
Lp norm of partial sums fluctuation about their conditional expectation

nipy.labs.utils.random_threshold.randthresh_varwind_gaussnull_connex(X, K, XYZ, p=inf, stop=False, verbose=False)

Random threshold with fixed-window and gaussian null distribution, using connexity constraint on non-null set.

Parameters X (n,): Observations (assumed Gaussian under H0):
XYZ (3,n): voxel coordinates:
K <int>:
Some positive integer (lower bound on the number of null hypotheses)
p <float>: Lp norm:
stop <bool>: Stop when minimum is attained (save computation time):

Returns C (n-K):
Lp norm of partial sums fluctuation about their conditional expectation

nipy.labs.utils.random_threshold.randthresh_varwind_knownull(X, K, p=inf, stop=False, verbose=False)

Random threshold with varying window and known null distribution

Parameters X (n,): Observations (Exp(1) under H0):
K <int>:
Some positive integer (lower bound on the number of null hypotheses)
p <float>: Lp norm:
stop <bool>: Stop when minimum is attained (save computation time):

Returns C (n-K):
Lp norm of partial sums fluctuation about their conditional expectation

nipy.labs.utils.random_threshold.randthresh_varwind_knownull_connex(X, K, XYZ, p=inf, stop=False, verbose=False)

Random threshold with varying window and known null distribution

Parameters X (n,): Observations (Exp(1) under H0):
K <int>:
Some positive integer (lower bound on the number of null hypotheses)
XYZ (3,n): voxel coordinates:

p <float>: lp norm:

stop <bool>: Stop when minimum is attained (save computation time):

Returns C (n-K):

Lp norm of partial sums fluctuation about their conditional expectation

nipy.labs.utils.random_threshold.test_stat(X, p=inf)

Test statistic of global null hypothesis that all observations have zero-mean

Parameters X (n,): X[j] = -log(1-F(|Y[j]|))

where F: cdf of |Y[j]| under null hypothesis (must be computed beforehand)

p: Lp norm (<= inf) to use for computing test statistic

Returns D <float> : test statistic
109.1 Module: labs.utils.reproducibility_measures

These are several functions for computing reproducibility measures. A use script should be appended soon on the repository.

In general this proceeds as follows: The dataset is subject to jackknife subampling (‘splitting’), each subsample being analysed independently. A reproducibility measure is then derived;

All is used to produce the work described in Analysis of a large fMRI cohort: Statistical and methodological issues for group analyses. Thirion B, Pinel P, Meriaux S, Roche A, Dehaene S, Poline JB. Neuroimage. 2007 Mar;35(1):105-20.
Bertrand Thirion, 2009-2010

109.2 Functions

nipy.labs.utils.reproducibility_measures.bootstrap_group(nsubj, ngroups)
  Split the proposed group into redundant subgroups by bootstrap

Parameters  
  nsubj (int) the number of subjects in the population :
  ngroups(int) Number of subbgroups to be drawn :

Returns  
  samples: a list of ngroups arrays containing :
  the indexes of the subjects in each subgroup

nipy.labs.utils.reproducibility_measures.cluster_reproducibility(data, vardata, doomain, ngroups, sigma, method='crfx', swap=False, verbose=0, **kwargs)

Returns a measure of cluster-level reproducibility of activation patterns (i.e. how far clusters are from each other)

Parameters  
  data: array of shape (nvox,nsubj) :
  the input data from which everything is computed
  vardata: array of shape (nvox,nsubj) :

715
the variance of the data that is also available

domain: referential- and domain- defining image instance :

ngroups (int), :

Number of subgroups to be drawn

sigma (float): parameter that encodes how far far is :

threshold (float): : binarization threshold

method='crfx', string to be chosen among ‘crfx’, ‘cmfx’ or ‘cffx’ :

inference method under study

swap = False: if True, a random sign swap of the data is performed :

This is used to simulate a null hypothesis on the data.

verbose=0 : verbosity mode

Returns score (float): the desired cluster-level reproducibility index :

nipy.labs.utils.reproducibility_measures.cluster_threshold(stat_map, domain, th, csize)

Perform a thresholding of a map at the cluster-level

Parameters stat_map: array of shape(nbvox) :

the input data

domain: Nifti1Image instance, :

referential- and domain-defining image

th (float): cluster-forming threshold :

csize (int>0): cluster size threshold :

Returns binary array of shape (nvox): the binarized thresholded map :

Notes

Should be replaced by a more standard function in the future

nipy.labs.utils.reproducibility_measures.conjunction(x, vx, k)

Returns a conjunction statistic as the sum of the k lowest t-values

Parameters x: array of shape(nrows, ncols), :

effect matrix

vx: array of shape(nrows, ncols), :

variance matrix

k: int, :

number of subjects in the conjunction

Returns t array of shape(nrows): conjunction statistic :
**nipy.labs.utils.reproducibility_measures.coord_bsa** *(domain, betas, theta=3.0, dmax=5.0, ths=0, thq=0.5, smin=0, afname=None)*

Main function for performing bsa on a dataset where bsa = nipy.labs.spatial_models.bayesian_structural_analysis

**Parameters**

- **domain**: image instance, referential- and domain-defining image
- **betas**: array of shape (nbnodes, subjects), the multi-subject statistical maps
- **theta**: float, optional, first level threshold
- **dmax**: float>0, optional, expected cluster std in the common space in units of coord
- **ths**: int, >=0, optional, representatitivity threshold
- **thq**: float, optional, posterior significance threshold should be in [0,1]
- **smin**: int, optional, minimal size of the regions to validate them
- **afname**: string, optional, path where intermediate resultts cam be pickelized

**Returns**

- **afcoord** array of shape(number_of_regions,3): coordinate of the found landmark regions

**nipy.labs.utils.reproducibility_measures.draw_samples** *(nsubj, ngroups, split_method='default')*

Draw randomly ngroups sets of samples from [0..nsubj-1]

**Parameters**

- **nsubj**, int, the total number of items
- **ngroups**, int, the number of desired groups
- **split_method**: string, optional, to be chosen among ‘default’, ‘bootstrap’, ‘jacknife’ if ‘bootstrap’, then each group will be nsubj
  - drawn with repetitions among nsubj
  - if ‘jacknife’ the population is divided into ngroups disjoint equally-sized subgroups
  - if ‘default’, ‘bootstrap’ is used when nsubj < 10 * ngroups otherwise jacknife is used

**Returns**

- **samples**, a list of ngroups array that represent the subsets
  - fixme: this should allow variable bootstrap,
  - i.e. draw ngroups of groupsize among nsubj
nipy.labs.utils.reproducibility_measures.fttest(x, vx)

Assuming that x and vx represent a effect and variance estimates, returns a cumulated ('fixed effects') t-test of the data over each row

Parameters  
  x: array of shape(nrows, ncols): effect matrix :  
  vx: array of shape(nrows, ncols): variance matrix :

Returns  
  t array of shape(nrows): fixed effect statistics array :

nipy.labs.utils.reproducibility_measures.get_cluster_position_from_thresholded_map(stat_map, domain, thr=3.0, csize=10)

the clusters above thr of size greater than csize in 18-connectivity are computed

Parameters  
  stat_map : array of shape (nbvox), 
    map to threshold 
  mask: Nifti1Image instance, : 
    referential- and domain-defining image 
  thr: float, optional, : 
    cluster-forming threshold 
  csize=10: int : 
    cluster size threshold 

Returns  
  positions array of shape(k,anat_dim): :  
    the cluster positions in physical coordinates where k= number of clusters if no such cluster exists, None is returned

nipy.labs.utils.reproducibility_measures.get_peak_position_from_thresholded_map(stat_map, domain, thr=3.0, csize=10)

The peaks above thr in 18-connectivity are computed

Parameters  
  stat_map: array of shape (nbvox): map to threshold :  
    domain: referential- and domain-defining image : 
  thr, float: cluster-forming threshold : 

Returns  
  positions array of shape(k,anat_dim): :  
    the cluster positions in physical coordinates where k= number of clusters if no such cluster exists, None is returned
nipy.labs.utils.reproducibility_measures.group_reproducibility_metrics(mask_images, contrast_images, variance_images, thresholds, ngroups, method, cluster_threshold=10, number_of_samples=10, sigma=6.0, do_clusters=True, do_voxels=True, do_peaks=True, swap=False)

Main function to perform reproducibility analysis, including nifti io

Parameters  threshold: list or 1-d array, :
the thresholds to be tested

Returns  cluster_rep_results: dictionary, :
results of cluster-level reproducibility analysis

voxel_rep_results: dictionary, :
results of voxel-level reproducibility analysis

peak_rep_results: dictionary, :
results of peak-level reproducibility analysis

nipy.labs.utils.reproducibility_measures.histo_repro(h)
Given the histogram h, compute a standardized reproducibility measure

Parameters  h array of shape(xmax+1), the histogram values :

Returns  hr, float: the measure :

nipy.labs.utils.reproducibility_measures.map_reproducibility(data, vardata, domain, ngroups, method='crfx', swap=False, verbose=0, **kwargs)

Return a reproducibility map for the given method

Parameters  data: array of shape (nvox,nsubj) :
the input data from which everything is computed

vardata: array of the same size :
the corresponding variance information

domain: referential- and domain-defining image :
ngroups (int): the size of each subgroup to be studied :

threshold (float): binarization threshold :
Inference method under study

Verbose=0 : verbosity mode

Returns rmap: array of shape(nvox):

the reproducibility map

nipy.labs.utils.reproducibility_measures.mfx_ttest(x, vx)

Idem fittest, but returns a mixed-effects statistic

Parameters x: array of shape(nrows, ncols): effect matrix:

vx: array of shape(nrows, ncols): variance matrix:

Returns t array of shape(nrows): mixed effect statistics array:

nipy.labs.utils.reproducibility_measures.peak_reproducibility(data, vardata, domain, ngroups, sigma, method='crfx', swap=False, verbose=0, **kwargs)

Return a measure of cluster-level reproducibility of activation patterns (i.e. how far clusters are from each other)

Parameters data: array of shape (nvox,nsubj):

the input data from which everything is computed

vardata: array of shape (nvox,nsubj):

the variance of the data that is also available

domain: referential- and domain-defining image:

ngroups (int),:

Number of subgroups to be drawn

sigma: float, parameter that encodes how far far is:

threshold: float, binarization threshold:

method: string to be chosen among ‘crfx’, ‘cmfx’ or ‘cffx’,:

inference method under study

swap = False: if True, a random sign swap of the data is performed:

This is used to simulate a null hypothesis on the data.

verbose=0 : verbosity mode

Returns score (float): the desired cluster-level reproducibility index:

nipy.labs.utils.reproducibility_measures.split_group(nsubj, ngroups)

Split the proposed group into random disjoint subgroups

Parameters nsubj (int) the number of subjects to be split:

ngroups(int) Number of subgroups to be drawn:

Returns samples: a list of ngroups arrays containing:
the indexes of the subjects in each subgroup

nipy.labs.utils.reproducibility_measures.statistics_from_position(target, data, sigma=1.0)

Return a number characterizing how close data is from target using a kernel-based statistic

Parameters  
  target: array of shape(nt,anat_dim) or None:
  the target positions
  data: array of shape(nd,anat_dim) or None:
  the data position
  sigma=1.0 (float), kernel parameter:
  or a distance that say how good good is

Returns  
  sensitivity (float): how well the targets are fitted:
  by the data in [0,1] interval 1 is good 0 is bad

nipy.labs.utils.reproducibility_measures.ttest(x)

Returns the t-test for each row of the data x

nipy.labs.utils.reproducibility_measures.voxel_reproducibility(data, vardata, do-

domain, ngroups, method='crfx', swap=False, verbose=0, **kwargs)

return a measure of voxel-level reproducibility of activation patterns

Parameters  
  data: array of shape (nvox,nsubj):
  the input data from which everything is computed
  vardata: array of shape (nvox,nsubj):
  the corresponding variance information ngroups (int): Number of subbgroups to be drawn
  domain: referential- and domain-defining image:
  ngroups: int:
  number of groups to be used in the resampling procedure
  method: string, to be chosen among ‘crfx’, ‘cmfx’, ‘cffx’:
  inference method under study
  verbose: bool, verbosity mode:

Returns  
  kappa (float): the desired reproducibility index:

nipy.labs.utils.reproducibility_measures.voxel_thresholded_ttest(x, threshold)

Returns a binary map of the ttest>threshold
110.1 Module: labs.utils.simul_multisubject_fmri_dataset

This module contains a function to produce a dataset which simulates a collection of 2D images. This dataset is saved as a 3D image (each slice being a subject) and a 3D array.

Author: Bertrand Thirion, 2008-2010

110.2 Functions

```
surrogate_2d_dataset(n_subj=10, shape=(30, 30), sk=1.0, noise_level=1.0, pos=array([[6, 7], [10, 10], [15, 10]]), ampli=array([3, 4, 4]), spatial_jitter=1.0, signal_jitter=1.0, width=5.0, width_jitter=0, out_text_file=None, out_image_file=None, seed=False)
```
Create surrogate (simulated) 2D activation data with spatial noise

Parameters

- **n_subj**: integer, optional
  - The number of subjects, i.e., the number of different maps generated.
- **shape=(30,30)**: tuple of integers
  - The shape of each image.
- **sk**: float, optional
  - Amount of spatial noise smoothness.
- **noise_level**: float, optional
  - Amplitude of the spatial noise. amplitude=noise_level
- **pos**: 2D ndarray of integers, optional
  - x, y positions of the various simulated activations.
- **ampli**: 1D ndarray of floats, optional
  - Respective amplitude of each activation.
- **spatial_jitter**: float, optional
  - Random spatial jitter added to the position of each activation, in pixel.
- **signal_jitter**: float, optional
  - Random amplitude fluctuation for each activation, added to the amplitude specified by ampli
- **width**: float or ndarray, optional
  -
Width of the activations

width_jitter: float :
    Relative width jitter of the blobs

out_text_file: string or None, optionnal :
    If not None, the resulting array is saved as a text file with the given file name

out_image_file: string or None, optionnal :
    If not None, the resulting is saved as a nifti file with the given file name.

seed=False: int, optionnal :
    If seed is not False, the random number generator is initialized at a certain value

Returns  dataset: 3D ndarray :
    The surrogate activation map, with dimensions (n_subj,) + shape

Create surrogate (simulated) 3D activation data with spatial noise.

Parameters  n_subj: integer, optionnal :
    The number of subjects, ie the number of different maps generated.

shape=(20,20,20): tuple of 3 integers, :
    the shape of each image

mask=None: Nifti1Image instance, :
    referential- and mask- defining image (overrides shape)

sk: float, optionnal :
    Amount of spatial noise smoothness.

noise_level: float, optionnal :
    Amplitude of the spatial noise. amplitude=noise_level)

pos: 2D ndarray of integers, optionnal :
    x, y positions of the various simulated activations.

ampli: 1D ndarray of floats, optionnal :
Respective amplitude of each activation

**spatial_jitter**: float, optional:
Random spatial jitter added to the position of each activation, in pixel.

**signal_jitter**: float, optional:
Random amplitude fluctuation for each activation, added to the amplitude specified by

**width**: float or ndarray, optional:
Width of the activations

**out_text_file**: string or None, optional:
If not None, the resulting array is saved as a text file with the given file name

**out_image_file**: string or None, optional:
If not None, the resulting is saved as a nifti file with the given file name.

**seed=False**: int, optional:
If seed is not False, the random number generator is initialized at a certain value

**Returns**

*dataset*: 3D ndarray:
The surrogate activation map, with dimensions \((n_{subj},) + \text{shape}\)

Create surrogate (simulated) 3D activation data with spatial noise.

**Parameters**

*shape = (20, 20, 20)*: tuple of integers, :
the shape of each image

*mask=None*: brifti image instance, :
referential- and mask- defining image (overrides shape)

*n_scans*: int, optional, :
number of scans to be simulated overridden by the design matrix

*n_sess*: int, optional, :
the number of simulated sessions

*dmtx*: array of shape\((n_{scans}, n_{rows})\), :
the design matrix

*sk*: float, optional :
Amount of spatial noise smoothness.
noise_level: float, optional:
    Amplitude of the spatial noise. amplitude=noise_level)

signal_level: float, optional:
    Amplitude of the signal

out_image_file: string or list of strings or None, optional:
    If not None, the resulting is saved as (set of) nifti file(s) with the given file path(s)

seed=False: int, optional:
    If seed is not False, the random number generator is initialized at a certain value

Returns dataset: a list of n_sess ndarray of shape:
    (shape[0], shape[1], shape[2], n_scans) The surrogate activation map
LABS.UTILS.ZSCORE

111.1 Module: labs.utils.zscore

nipy.labs.utils.zscore.zscore(pvalue)

Return the z-score corresponding to a given p-value.
LABS.VIZ_TOOLS.ACTIVATION_MAPS

112.1 Module: labs.viz_tools.activation_maps

Functions to do automatic visualization of activation-like maps.

For 2D-only visualization, only matplotlib is required. For 3D visualization, Mayavi, version 3.0 or greater, is required.

For a demo, see the 'demo_plot_map' function.

112.2 Functions

nipy.labs.viz_tools.activation_maps.demo_plot_map(do3d=False, **kwargs)

Demo activation map plotting.

nipy.labs.viz_tools.activation_maps.plot_anat(anat=None, anat_affine=None, cut_coords=None, slicer='ortho', figure=None, axes=None, title=None, annotate=True, draw_cross=True, black_bg=False, dim=False, cmap=<matplotlib.colors.LinearSegmentedColormap instance at 0x311e248>)

Plot three cuts of an anatomical image (Frontal, Axial, and Lateral)

Parameters

- **anat**: 3D ndarray, optional
  
The anatomical image to be used as a background. If None is given, nipy tries to find a T1 template.

- **anat_affine**: 4x4 ndarray, optional
  
The affine matrix going from the anatomical image voxel space to MNI space. This parameter is not used when the default anatomical is used, but it is compulsory when using an explicit anatomical image.

- **figure**: integer or matplotlib figure, optional
  
Matplotlib figure used or its number. If None is given, a new figure is created.

- **cut_coords**: None, or a tuple of floats
  
The MNI coordinates of the point where the cut is performed, in MNI coordinates and order. If slicer is 'ortho', this should be a 3-tuple: (x, y, z) For slicer == 'x', 'y', or 'z', then these are the coordinates of each cut in the corresponding direction. If None is given, the cuts is calculated automatically.
slicer: \{‘ortho’, ‘x’, ‘y’, ‘z’\}:

Choose the direction of the cuts. With ‘ortho’ three cuts are performed in orthogonal
directions.

figure : integer or matplotlib figure, optional

Matplotlib figure used or its number. If None is given, a new figure is created.

axes : matplotlib axes or 4 tuple of float: (xmin, ymin, width, height), optional

The axes, or the coordinates, in matplotlib figure space, of the axes used to display the
plot. If None, the complete figure is used.

title : string, optional

The title displayed on the figure.

annotate: boolean, optional:

If annotate is True, positions and left/right annotation are added to the plot.

draw_cross: boolean, optional:

If draw_cross is True, a cross is drawn on the plot to indicate the cut position.

black_bg: boolean, optional:

If True, the background of the image is set to be black. If you wish to save figures with
a black background, you will need to pass “facecolor=’k’, edgecolor=’k’” to pylab’s
savefig.

cmap: matplotlib colormap, optional:

The colormap for the anat

Notes

Arrays should be passed in numpy convention: (x, y, z) ordered.

Plot three cuts of a given activation map (Frontal, Axial, and Lateral)

Parameters

map : 3D ndarray

The activation map, as a 3D image.

affine : 4x4 ndarray

The affine matrix going from image voxel space to MNI space.

cut_coords: None, or a tuple of floats:

The MNI coordinates of the point where the cut is performed, in MNI coordinates and
order. If slicer is ‘ortho’, this should be a 3-tuple: (x, y, z) For slicer == ‘x’, ‘y’, or
‘z’, then these are the coordinates of each cut in the corresponding direction. If None is
given, the cuts is calculated automatically.
anat : 3D ndarray or False, optional

The anatomical image to be used as a background. If None, the MNI152 T1 1mm template is used. If False, no anat is displayed.

anat_affine : 4x4 ndarray, optional

The affine matrix going from the anatomical image voxel space to MNI space. This parameter is not used when the default anatomical is used, but it is compulsory when using an explicit anatomical image.

slicer: {'ortho', 'x', 'y', 'z'} :

Choose the direction of the cuts. With 'ortho' three cuts are performed in orthogonal directions

figure : integer or matplotlib figure, optional

Matplotlib figure used or its number. If None is given, a new figure is created.

axes : matplotlib axes or 4 tuple of float: (xmin, ymin, width, height), optional

The axes, or the coordinates, in matplotlib figure space, of the axes used to display the plot. If None, the complete figure is used.

title : string, optional

The title displayed on the figure.

threshold : a number, None, or 'auto'

If None is given, the maps are not thresholded. If a number is given, it is used to threshold the maps: values below the threshold are plotted as transparent. If auto is given, the threshold is determined magically by analysis of the map.

annotate: boolean, optional :

If annotate is True, positions and left/right annotation are added to the plot.

draw_cross: boolean, optional :

If draw_cross is True, a cross is drawn on the plot to indicate the cut position.

do3d: {True, False or 'interactive'}, optional :

If True, Mayavi is used to plot a 3D view of the map in addition to the slicing. If 'interactive', the 3D visualization is displayed in an additional interactive window.

threshold_3d: :

The threshold to use for the 3D view (if any). Defaults to the same threshold as that used for the 2D view.

view_3d: tuple, :

The view used to take the screenshot: azimuth, elevation, distance and focalpoint, see the docstring of mlab.view.

black_bg: boolean, optional :

If True, the background of the image is set to be black. If you wish to save figures with a black background, you will need to pass “facecolor='k’, edgecolor='k’” to pylab’s savefig.

kwargs: extra keyword arguments, optional :

Extra keyword arguments passed to pylab.imshow
Notes

Arrays should be passed in numpy convention: (x, y, z) ordered.

Use masked arrays to create transparency:

    import numpy as np
    map = np.ma.masked_less(map, 0.5)
    plot_map(map, affine)
113.1 Module: labs.viz_tools.anat_cache

3D visualization of activation maps using Mayavi

nipy.labs.viz_tools.anat_cache.find_mni_template()

Try to find an MNI template on the disk.
114.1 Module: labs.viz_tools.cm

Matplotlib colormaps useful for neuroimaging.

114.2 Functions

```python
nipy.labs.viz_tools.cm.alpha_cmap(color, name='')
```
Return a colormap with the given color, and alpha going from zero to 1.

**Parameters**
- **color**: (r, g, b), or a string
  A triplet of floats ranging from 0 to 1, or a matplotlib color string

```python
nipy.labs.viz_tools.cm.dim_cmap(cmap, factor=0.3, to_white=True)
```
Dim a colormap to white, or to black.

```python
nipy.labs.viz_tools.cm.replace_inside(outer_cmap, inner_cmap, vmin, vmax)
```
Replace a colormap by another inside a pair of values.
115.1 Module: labs.viz_tools.coord_tools

Misc tools to find activations and cut on maps

115.2 Functions

nipy.labs.viz_tools.coord_tools.coord_transform(x, y, z, affine)
Convert the x, y, z coordinates from one image space to another space.

Parameters
- x: number or ndarray
  The x coordinates in the input space
- y: number or ndarray
  The y coordinates in the input space
- z: number or ndarray
  The z coordinates in the input space
- affine: 2D 4x4 ndarray
  affine that maps from input to output space.

Returns
- x: number or ndarray
  The x coordinates in the output space
- y: number or ndarray
  The y coordinates in the output space
- z: number or ndarray
  The z coordinates in the output space

Warning: The x, y and z have their Talairach ordering, not 3D:
numy image ordering.

nipy.labs.viz_tools.coord_tools.find_cut_coords(map, mask=None, activation_threshold=None)
Find the center of the largest activation connect component.

Parameters
- map: 3D ndarray
The activation map, as a 3D image.

**mask** : 3D ndarray, boolean, optional

An optional brain mask.

**activation_threshold** : float, optional

The lower threshold to the positive activation. If None, the activation threshold is computed using find_activation.

**Returns**

- **x**: float
  - the x coordinate in voxels.
- **y**: float
  - the y coordinate in voxels.
- **z**: float
  - the z coordinate in voxels.

```python
nipy.labs.viz_tools.coord_tools.get_mask_bounds(mask, affine)
```

Return the world-space bounds occupied by a mask given an affine.

**Notes**

- The mask should have only one connect component.
- The affine should be diagonal or diagonal-permuted.
LABS.VIZTOOLS.MAPS_3D

116.1 Module: labs.viz_tools.maps_3d

3D visualization of activation maps using Mayavi

116.2 Functions

nipy.labs.viz_tools.maps_3d.affine_img_src(data, affine, scale=1, name='AffineImage', reverse_x=False)

Make a Mayavi source defined by a 3D array and an affine, for which the voxel of the 3D array are mapped by the affine.

Parameters

- **data**: 3D ndarray
  - The data arrays
- **affine**: (4 x 4) ndarray
  - The (4 x 4) affine matrix relating voxels to world coordinates.
- **scale**: float, optional
  - An optional addition scaling factor.
- **name**: string, optional
  - The name of the Mayavi source created.
- **reverse_x**: boolean, optional
  - Reverse the x (lateral) axis. Useful to compared with images in radiologic convention.

Notes

The affine should be diagonal.

nipy.labs.viz_tools.maps_3d.autocrop_img(img, bg_color)
nipy.labs.viz_tools.maps_3d.demo_plot_map_3d()
nipy.labs.viz_tools.maps_3d.m2screenshot(mayavi_fig=None, mpl_axes=None, autocrop=True)

Capture a screenshot of the Mayavi figure and display it in the matplotlib axes.
3D anatomical display

**Parameters**

- **skull_percentile** : float, optional
  
  The percentile of the values in the image that delimit the skull from the outside of the brain. The smaller the fraction of your field of view is occupied by the brain, the larger this value should be.

- **wm_percentile** : float, optional
  
  The percentile of the values in the image that delimit the white matter from the grey matter. Typically, this is skull_percentile + 1

Plot a 3D volume rendering view of the activation, with an outline of the brain.

**Parameters**

- **map** : 3D ndarray
  
  The activation map, as a 3D image.

- **affine** : 4x4 ndarray
  
  The affine matrix going from image voxel space to MNI space.

- **cut_coords** : 3-tuple of floats, optional
  
  The MNI coordinates of a 3D cursor to indicate a feature or a cut, in MNI coordinates and order.

- **anat** : 3D ndarray, optional
  
  The anatomical image to be used as a background. If None, the MNI152 T1 1mm template is used. If False, no anatomical image is used.

- **anat_affine** : 4x4 ndarray, optional
  
  The affine matrix going from the anatomical image voxel space to MNI space. This parameter is not used when the default anatomical is used, but it is compulsory when using an explicit anatomical image.

- **threshold** : float, optional
  
  The lower threshold of the positive activation. This parameter is used to threshold the activation map.

- **offscreen** : boolean, optional
  
  If True, Mayavi attempts to plot offscreen. Will work only with VTK >= 5.2.

- **vmin** : float, optional
  
  The minimal value, for the colormap.

- **vmax** : float, optional
  
  The maximum value, for the colormap.

- **cmap** : a callable, or a pylab colormap
A callable returning a (n, 4) array for n values between 0 and 1 for the colors. This can be for instance a pylab colormap.

**Notes**

If you are using a VTK version below 5.2, there is no way to avoid opening a window during the rendering under Linux. This is necessary to use the graphics card for the rendering. You must maintain this window on top of others and on the screen.
117.1 Module: labs.viz_tools.slicers

The Slicer classes.

The main purpose of these classes is to have auto adjust of axes size to the data with different layout of cuts.

117.2 Classes

117.2.1 BaseSlicer

class nipy.labs.viz_tools.slicers.BaseSlicer(cut_coords, axes=None, black_bg=False)  
Bases: object

The main purpose of these class is to have auto adjust of axes size to the data with different layout of cuts.

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotate</td>
<td>Add annotations to the plot.</td>
</tr>
<tr>
<td>contour_map</td>
<td>Contour a 3D map in all the views.</td>
</tr>
<tr>
<td>edge_map</td>
<td>Plot the edges of a 3D map in all the views.</td>
</tr>
<tr>
<td>find_cut_coords</td>
<td></td>
</tr>
</tbody>
</table>

Continued on next page
Table 117.1 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>init_with_figure</code></td>
<td>([data, affine, threshold, ...])</td>
</tr>
<tr>
<td><code>plot_map</code></td>
<td>(map, affine[, threshold])</td>
</tr>
<tr>
<td></td>
<td>Plot a 3D map in all the views.</td>
</tr>
<tr>
<td><code>title</code></td>
<td>(text[, x, y, size, color, bgcolor, alpha])</td>
</tr>
<tr>
<td></td>
<td>Write a title to the view.</td>
</tr>
</tbody>
</table>

__init__ (cut_coords, axes=None, black_bg=False)
Create 3 linked axes for plotting orthogonal cuts.

**Parameters**
- **cut_coords**: 3 tuple of ints
  
  The cut position, in world space.

- **axes**: matplotlib axes object, optional
  
  The axes that will be subdivided in 3.

- **black_bg**: boolean, optional
  
  If True, the background of the figure will be put to black. If you wish to save figures with a black background, you will need to pass `"facecolor='k', edgecolor='k'"` to `pylab`'s `savefig`.

annotate (left_right=True, positions=True, size=12, **kwargs)
Add annotations to the plot.

**Parameters**
- **left_right**: boolean, optional
  
  If left_right is True, annotations indicating which side is left and which side is right are drawn.

- **positions**: boolean, optional
  
  If positions is True, annotations indicating the positions of the cuts are drawn.

- **size**: integer, optional
  
  The size of the text used.

- **kwargs**
  
  Extra keyword arguments are passed to matplotlib's text function.

contour_map (map, affine, **kwargs)
Contour a 3D map in all the views.

**Parameters**
- **map**: 3D ndarray
  
  The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

- **affine**: 4x4 ndarray
  
  The affine matrix giving the transformation from voxel indices to world space.

- **kwargs**
  
  Extra keyword arguments are passed to contour.

draw_edge (map, affine, color='r')
Plot the edges of a 3D map in all the views.

**Parameters**
- **map**: 3D ndarray
  
  The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.
affine: 4x4 ndarray:
The affine matrix giving the transformation from voxel indices to world space.

color: matplotlib color: string or (r, g, b) value:
The color used to display the edge map

static find_cut_coords(data=None, affine=None, threshold=None, cut_coords=None):

classmethod init_with_figure(data=None, affine=None, threshold=None, cut_coords=None, figure=None, axes=None, black_bg=False, leave_space=False):

plot_map(map, affine, threshold=None, **kwargs):
Plot a 3D map in all the views.

Parameters
  map: 3D ndarray:
The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

affine: 4x4 ndarray:
The affine matrix giving the transformation from voxel indices to world space.

threshold: a number, None, or ‘auto’
If None is given, the maps are not thresholded. If a number is given, it is used to threshold the maps: values below the threshold are plotted as transparent.

kwargs:
Extra keyword arguments are passed to imshow.

title(text, x=0.01, y=0.99, size=15, color=None, bgcolor=None, alpha=1, **kwargs):
Write a title to the view.

Parameters
  text: string:
The text of the title

x: float, optional:
The horizontal position of the title on the frame in fraction of the frame width.

y: float, optional:
The vertical position of the title on the frame in fraction of the frame height.

size: integer, optional:
The size of the title text.

color: matplotlib color specifier, optional:
The color of the font of the title.

bgcolor: matplotlib color specifier, optional:
The color of the background of the title.

alpha: float, optional:
The alpha value for the background.

kwargs:
Extra keyword arguments are passed to matplotlib’s text function.
117.2.2 BaseStackedSlicer

class nipy.labs.viz_tools.slicers.BaseStackedSlicer(cut_coords, axes=None, black_bg=False)
  
  Bases: nipy.labs.viz_tools.slicers.BaseSlicer

A class to create linked axes for plotting stacked cuts of 3D maps.

Notes

The extent of the different axes are adjusted to fit the data best in the viewing area.

Attributes

<table>
<thead>
<tr>
<th>axes: dictionary of axes</th>
<th>The axes used to plot each view.</th>
</tr>
</thead>
<tbody>
<tr>
<td>frame_axes: axes</td>
<td>The axes framing the whole set of views.</td>
</tr>
</tbody>
</table>

Methods

- **annotate**([left_right, positions, size])  Add annotations to the plot.
- **contour_map**(map, affine, **kwargs)  Contour a 3D map in all the views.
- **draw_cross**([cut_coords])  Draw a crossbar on the plot to show where the cut is performed.
- **edge_map**(map, affine[, color])  Plot the edges of a 3D map in all the views.
- **find_cut_coords**([data, affine, threshold, ...])  Plot a 3D map in all the views.
- **init_with_figure**([data, affine, threshold, ...])  Plot a 3D map in all the views.
- **plot_map**(map, affine[, threshold])  Write a title to the view.

__init__ (cut_coords, axes=None, black_bg=False)

Create 3 linked axes for plotting orthogonal cuts.

**Parameters**

- **cut_coords**: 3 tuple of ints
  
The cut position, in world space.

- **axes**: matplotlib axes object, optional
  
The axes that will be subdivided in 3.

- **black_bg**: boolean, optional
  
  If True, the background of the figure will be put to black. If you whish to save figures with a black background, you will need to pass “facecolor=’k’, edgecolor=’k’” to pylab’s savefig.

**annotate** (left_right=True, positions=True, size=12, **kwargs)

Add annotations to the plot.

**Parameters**

- **left_right**: boolean, optional
  
  If left_right is True, annotations indicating which side is left and which side is right are drawn.

- **positions**: boolean, optional
If positions is True, annotations indicating the positions of the cuts are drawn.

size: integer, optional:
The size of the text used.

kwarg: :
Extra keyword arguments are passed to matplotlib’s text function.

**contour_map**(map, affine, **kwargs)
Contour a 3D map in all the views.

**Parameters**

- **map**: 3D ndarray:
The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

- **affine**: 4x4 ndarray:
The affine matrix giving the transformation from voxel indices to world space.

- **kwarg**: :
Extra keyword arguments are passed to contour.

**draw_cross**(cut_coords=None, **kwargs)
Draw a crossbar on the plot to show where the cut is performed.

**Parameters**

- **cut_coords**: 3-tuple of floats, optional:
The position of the cross to draw. If none is passed, the ortho_slicer’s cut coordinates are used.

- **kwarg**: :
Extra keyword arguments are passed to axhline

**edge_map**(map, affine, color='r')
Plot the edges of a 3D map in all the views.

**Parameters**

- **map**: 3D ndarray:
The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

- **affine**: 4x4 ndarray:
The affine matrix giving the transformation from voxel indices to world space.

- **color**: matplotlib color: string or (r, g, b) value:
The color used to display the edge map

**classmethod** find_cut_coords**(data=None, affine=None, threshold=None, cut_coords=None)**

**classmethod** init_with_figure**(data=None, affine=None, threshold=None, cut_coords=None, figure=None, axes=None, black_bg=False, leave_space=False)**

**plot_map**(map, affine, threshold=None, **kwargs)
Plot a 3D map in all the views.

**Parameters**

- **map**: 3D ndarray:
The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

- **affine**: 4x4 ndarray:
The affine matrix giving the transformation from voxel indices to world space.

**threshold**: a number, None, or ‘auto’

If None is given, the maps are not thresholded. If a number is given, it is used to threshold the maps: values below the threshold are plotted as transparent.

**kwargs**: 
Extra keyword arguments are passed to imshow.

**title** *(text, x=0.01, y=0.99, size=15, color=None, bgcolor=None, alpha=1, **kwargs)*

Write a title to the view.

**Parameters**
- **text**: string
  - The text of the title
- **x**: float, optional
  - The horizontal position of the title on the frame in fraction of the frame width.
- **y**: float, optional
  - The vertical position of the title on the frame in fraction of the frame height.
- **size**: integer, optional
  - The size of the title text.
- **color**: matplotlib color specifier, optional
  - The color of the font of the title.
- **bgcolor**: matplotlib color specifier, optional
  - The color of the background of the title.
- **alpha**: float, optional
  - The alpha value for the background.
- **kwargs**: 
  - Extra keyword arguments are passed to matplotlib’s text function.

### 117.2.3 CutAxes

class *nipy.labs.viz_tools.slicers.CutAxes*(ax, direction, coord)

Bases: *object*

An MPL axis-like object that displays a cut of 3D volumes

**Methods**

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>do_cut(map, affine)</td>
<td>Cut the 3D volume into a 2D slice</td>
</tr>
<tr>
<td>draw_cut(cut, data_bounds, bounding_box[, type])</td>
<td></td>
</tr>
<tr>
<td>draw_left_right(size, bg_color, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>draw_position(size, bg_color, **kwargs)</td>
<td></td>
</tr>
<tr>
<td>get_object_bounds()</td>
<td>Return the bounds of the objects on this axes.</td>
</tr>
</tbody>
</table>
__init__ \((ax, direction, coord)\)

An MPL axis-like object that displays a cut of 3D volumes

Parameters

- **ax**: a MPL axes instance:
  The axes in which the plots will be drawn
- **direction**: \{'x', 'y', 'z'\}:
  The directions of the cut
- **coord**: float:
  The coordinate along the direction of the cut

do_cut \((map, affine)\)

Cut the 3D volume into a 2D slice

Parameters

- **map**: 3D ndarray:
  The 3D volume to cut
- **affine**: 4x4 ndarray:
  The affine of the volume
draw_cut \((cut, data_bounds, bounding_box, type='imshow', **kwargs)\)
draw_left_right \((size, bg_color, **kwargs)\)
draw_position \((size, bg_color, **kwargs)\)
get_object_bounds ()

    Return the bounds of the objects on this axes.

117.2.4 OrthoSlicer

class `nipy.labs.viz_tools.slicers.OrthoSlicer` \((cut_coords, axes=None, black_bg=False)\)

Bases: `nipy.labs.viz_tools.slicers.BaseSlicer`

A class to create 3 linked axes for plotting orthogonal cuts of 3D maps.

Notes

The extent of the different axes are adjusted to fit the data best in the viewing area.

Attributes

- **axes**: dictionary of axes
  The 3 axes used to plot each view.
- **frame_axes**: axes
  The axes framing the whole set of views.

Methods

- **annotate**\([left_right, positions, size]\)
  Add annotations to the plot.
- **contour_map**\((map, affine, **kwags)\)
  Contour a 3D map in all the views.
- **draw_cross**\([cut_coords]\)
  Draw a crossbar on the plot to show where the cut is performed.
Table 117.4 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>edge_map</td>
<td>Plot the edges of a 3D map in all the views.</td>
</tr>
<tr>
<td>find_cut_coords</td>
<td></td>
</tr>
<tr>
<td>init_with_figure</td>
<td></td>
</tr>
<tr>
<td>plot_map</td>
<td>Plot a 3D map in all the views.</td>
</tr>
<tr>
<td>title</td>
<td>Write a title to the view.</td>
</tr>
</tbody>
</table>

__init__ (cut_coords, axes=None, black_bg=False)
Create 3 linked axes for plotting orthogonal cuts.

Parameters
cut_coords: 3 tuple of ints:
The cut position, in world space.

axes: matplotlib axes object, optional:
The axes that will be subdivided in 3.

black_bg: boolean, optional:
If True, the background of the figure will be put to black. If you wish to save figures with a black background, you will need to pass “facecolor='k’, edgecolor='k’” to pylab’s savefig.

annotate (left_right=True, positions=True, size=12, **kwargs)
Add annotations to the plot.

Parameters
left_right: boolean, optional:
If left_right is True, annotations indicating which side is left and which side is right are drawn.

positions: boolean, optional:
If positions is True, annotations indicating the positions of the cuts are drawn.

size: integer, optional:
The size of the text used.

kwargs: :
Extra keyword arguments are passed to matplotlib’s text function.

contour_map (map, affine, **kwargs)
Contour a 3D map in all the views.

Parameters
map: 3D ndarray:
The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

affine: 4x4 ndarray:
The affine matrix giving the transformation from voxel indices to world space.

kwargs: :
Extra keyword arguments are passed to contour.

draw_cross (cut_coords=None, **kwargs)
Draw a crossbar on the plot to show where the cut is performed.

Parameters
cut_coords: 3-tuple of floats, optional:
The position of the cross to draw. If none is passed, the ortho_slicer’s cut coordinates are used.

**kwargs:
Extra keyword arguments are passed to axhline

def edge_map(map, affine, color='r')
Plot the edges of a 3D map in all the views.

Parameters
- **map**: 3D ndarray
  - The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.
- **affine**: 4x4 ndarray
  - The affine matrix giving the transformation from voxel indices to world space.
- **color**: matplotlib color: string or (r, g, b) value
  - The color used to display the edge map

static find_cut_coords(data=None, affine=None, threshold=None, cut_coords=None)

classmethod init_with_figure(data=None, affine=None, threshold=None, cut_coords=None, figure=None, axes=None, black_bg=False, leave_space=False)

def plot_map(map, affine, threshold=None, **kwargs)
Plot a 3D map in all the views.

Parameters
- **map**: 3D ndarray
  - The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.
- **affine**: 4x4 ndarray
  - The affine matrix giving the transformation from voxel indices to world space.
- **threshold**: a number, None, or ‘auto’
  - If None is given, the maps are not thresholded. If a number is given, it is used to threshold the maps: values below the threshold are plotted as transparent.
- **kwargs**:
  - Extra keyword arguments are passed to imshow.

def title(text, x=0.01, y=0.99, size=15, color=None, bgcolor=None, alpha=1, **kwargs)
Write a title to the view.

Parameters
- **text**: string
  - The text of the title
- **x**: float, optional
  - The horizontal position of the title on the frame in fraction of the frame width.
- **y**: float, optional
  - The vertical position of the title on the frame in fraction of the frame height.
- **size**: integer, optional
  - The size of the title text.
- **color**: matplotlib color specifier, optional
  - The color of the title text.
The color of the font of the title.

**bgcolor**: matplotlib color specifier, optional:

The color of the background of the title.

**alpha**: float, optional:

The alpha value for the background.

**kwargs**: Extra keyword arguments are passed to matplotlib's text function.

## 117.2.5 XSlicer

class nipy.labs.viz_tools.slicers.XSlicer

```python
Xslicer (cut_coords, axes=None, black_bg=False)
```

**Bases**: nipy.labs.viz_tools.slicers.BaseStackedSlicer

### Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>annotate</strong>(left_right, positions, size)</td>
<td>Add annotations to the plot.</td>
</tr>
<tr>
<td><strong>contour_map</strong>(map, affine, **kwargs)</td>
<td>Contour a 3D map in all the views.</td>
</tr>
<tr>
<td><strong>draw_cross</strong>(cut_coords)</td>
<td>Draw a crossbar on the plot to show where the cut is performed.</td>
</tr>
<tr>
<td><strong>edge_map</strong>(map, affine[, color])</td>
<td>Plot the edges of a 3D map in all the views.</td>
</tr>
<tr>
<td><strong>find_cut_coords</strong>([data, affine, threshold, ...])</td>
<td>Plot a 3D map in all the views.</td>
</tr>
<tr>
<td><strong>init_with_figure</strong>([data, affine, threshold, ...])</td>
<td>Plot a 3D map in all the views.</td>
</tr>
<tr>
<td><strong>plot_map</strong>(map, affine[, threshold])</td>
<td>Write a title to the view.</td>
</tr>
<tr>
<td><strong>title</strong>(text[, x, y, size, color, bgcolor, alpha])</td>
<td>Write a title to the view.</td>
</tr>
</tbody>
</table>

### __init__**(cut_coords, axes=None, black_bg=False)

Create 3 linked axes for plotting orthogonal cuts.

**Parameters**

- **cut_coords**: 3 tuple of ints:
  - The cut position, in world space.

- **axes**: matplotlib axes object, optional:
  - The axes that will be subdivided in 3.

- **black_bg**: boolean, optional:
  - If True, the background of the figure will be put to black. If you wish to save figures with a black background, you will need to pass “facecolor='k', edgecolor='k'” to `pylab`'s savefig.

### annotate**(left_right=True, positions= True, size= 12, **kwargs)

Add annotations to the plot.

**Parameters**

- **left_right**: boolean, optional:
  - If left_right is True, annotations indicating which side is left and which side is right are drawn.

- **positions**: boolean, optional:
  - If positions is True, annotations indicating the positions of the cuts are drawn.
size: integer, optional :
    The size of the text used.

kwargs:
    Extra keyword arguments are passed to matplotlib’s text function.

contour_map (map, affine, **kwargs)
    Contour a 3D map in all the views.

    Parameters
    map: 3D ndarray :
        The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.
    affine: 4x4 ndarray :
        The affine matrix giving the transformation from voxel indices to world space.
    kwargs:
        Extra keyword arguments are passed to contour.

draw_cross (cut_coords=None, **kwargs)
    Draw a crossbar on the plot to show where the cut is performed.

    Parameters
    cut_coords: 3-tuple of floats, optional :
        The position of the cross to draw. If none is passed, the ortho_slicer’s cut coordinates are used.
    kwargs:
        Extra keyword arguments are passed to axhline

draw_edge_map (map, affine, color='r')
    Plot the edges of a 3D map in all the views.

    Parameters
    map: 3D ndarray :
        The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.
    affine: 4x4 ndarray :
        The affine matrix giving the transformation from voxel indices to world space.
    color: matplotlib color: string or (r, g, b) value :
        The color used to display the edge map

classmethod find_cut_coords (data=None, affine=None, threshold=None, cut_coords=None)

classmethod init_with_figure (data=None, affine=None, threshold=None, cut_coords=None, figure=None, axes=None, black_bg=False, leave_space=False)

plot_map (map, affine, threshold=None, **kwargs)
    Plot a 3D map in all the views.

    Parameters
    map: 3D ndarray :
        The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.
    affine: 4x4 ndarray :
        The affine matrix giving the transformation from voxel indices to world space.
threshold : a number, None, or ‘auto’

If None is given, the maps are not thresholded. If a number is given, it is used to
threshold the maps: values below the threshold are plotted as transparent.

kwargs : 
Extra keyword arguments are passed to imshow.

title (text, x=0.01, y=0.99, size=15, color=None, bgcolor=None, alpha=1, **kwargs)
Write a title to the view.

Parameters

text: string :
The text of the title

x: float, optional :
The horizontal position of the title on the frame in fraction of the frame width.

y: float, optional :
The vertical position of the title on the frame in fraction of the frame height.

size: integer, optional :
The size of the title text.

color: matplotlib color specifier, optional :
The color of the font of the title.

bgcolor: matplotlib color specifier, optional :
The color of the background of the title.

alpha: float, optional :
The alpha value for the background.

kwargs: :
Extra keyword arguments are passed to matplotlib’s text function.

117.2.6 YSlicer

class nipy.labs.viz_tools.slicers.YSlicer (cut_coords, axes=None, black_bg=False)
Bases: nipy.labs.viz_tools.slicers.BaseStackedSlicer

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotate([left_right, positions, size])</td>
<td>Add annotations to the plot.</td>
</tr>
<tr>
<td>contour_map(map, affine, **kwargs)</td>
<td>Contour a 3D map in all the views.</td>
</tr>
<tr>
<td>draw_cross([cut_coords])</td>
<td>Draw a crossbar on the plot to show where the cut is performed.</td>
</tr>
<tr>
<td>edge_map(map, affine[, color])</td>
<td>Plot the edges of a 3D map in all the views.</td>
</tr>
<tr>
<td>find_cut_coords([data, affine, threshold, ...])</td>
<td></td>
</tr>
<tr>
<td>init_with_figure([data, affine, threshold, ...])</td>
<td></td>
</tr>
<tr>
<td>plot_map(map, affine[, threshold])</td>
<td>Plot a 3D map in all the views.</td>
</tr>
<tr>
<td>title(text[, x, y, size, color, bgcolor, alpha])</td>
<td>Write a title to the view.</td>
</tr>
</tbody>
</table>
__init__(cut_coords, axes=None, black_bg=False)
Create 3 linked axes for plotting orthogonal cuts.

Parameters  
cut_coords: 3 tuple of ints :
The cut position, in world space.

axes: matplotlib axes object, optional :
The axes that will be subdivided in 3.

black_bg: boolean, optional :
If True, the background of the figure will be put to black. If you whish to save figures with a black background, you will need to pass “facecolor='k', edgecolor='k'” to `pylab's savefig`.

annotate(left_right=True, positions=True, size=12, **kwargs)
Add annotations to the plot.

Parameters  
left_right: boolean, optional :
If left_right is True, annotations indicating which side is left and which side is right are drawn.

positions: boolean, optional :
If positions is True, annotations indicating the positions of the cuts are drawn.

size: integer, optional :
The size of the text used.

kwargs:
Extra keyword arguments are passed to `matplotlib's text function`.

contour_map(map, affine, **kwargs)
Contour a 3D map in all the views.

Parameters  
map: 3D ndarray :
The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

affine: 4x4 ndarray :
The affine matrix giving the transformation from voxel indices to world space.

kwargs:
Extra keyword arguments are passed to `contour`.

draw_cross(cut_coords=None, **kwargs)
Draw a crossbar on the plot to show where the cut is performed.

Parameters  
cut_coords: 3-tuple of floats, optional :
The position of the cross to draw. If none is passed, the `ortho_slicer`'s cut coordinates are used.

kwargs:
Extra keyword arguments are passed to `axhline`

draw_map(map, affine, color='r')
Plot the edges of a 3D map in all the views.
Parameters  

map: 3D ndarray:

The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

affine: 4x4 ndarray:

The affine matrix giving the transformation from voxel indices to world space.

color: matplotlib color: string or (r, g, b) value:

The color used to display the edge map

classmethod find_cut_coords(data=None, affine=None, threshold=None, cut_coords=None)

classmethod init_with_figure(data=None, affine=None, threshold=None, cut_coords=None, figure=None, axes=None, black_bg=False, leave_space=False)

plot_map(map, affine, threshold=None, **kwargs)

Plot a 3D map in all the views.

Parameters  

map: 3D ndarray:

The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

affine: 4x4 ndarray:

The affine matrix giving the transformation from voxel indices to world space.

threshold:

If None is given, the maps are not thresholded. If a number is given, it is used to threshold the maps: values below the threshold are plotted as transparent.

kwargs:

Extra keyword arguments are passed to imshow.

title(text, x=0.01, y=0.99, size=15, color=None, bgcolor=None, alpha=1, **kwargs)

Write a title to the view.

Parameters  

text: string:

The text of the title

x: float, optional:

The horizontal position of the title on the frame in fraction of the frame width.

y: float, optional:

The vertical position of the title on the frame in fraction of the frame height.

size: integer, optional:

The size of the title text.

color: matplotlib color specifier, optional:

The color of the font of the title.

bgcolor: matplotlib color specifier, optional:

The color of the background of the title.

alpha: float, optional:

The alpha value for the background.
kwargs:

Extra keyword arguments are passed to matplotlib’s text function.

117.2.7 ZSlicer

class nipy.labs.viz_tools.slicers.ZSlicer(cut_coords, axes=None, black_bg=False)
    Bases: nipy.labs.viz_tools.slicers.BaseStackedSlicer

Methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>annotate(left_right, positions, size)</td>
<td>Add annotations to the plot.</td>
</tr>
<tr>
<td>contour_map(map, affine, **kwargs)</td>
<td>Contour a 3D map in all the views.</td>
</tr>
<tr>
<td>draw_cross([cut_coords])</td>
<td>Draw a crossbar on the plot to show where the cut is performed.</td>
</tr>
<tr>
<td>edge_map(map, affine[, color])</td>
<td>Plot the edges of a 3D map in all the views.</td>
</tr>
<tr>
<td>find_cut_coords([data, affine, threshold, ...])</td>
<td>Find the cut coordinates for a 3D map.</td>
</tr>
<tr>
<td>init_with_figure([data, affine, threshold, ...])</td>
<td>Initialize the figure with data and affine.</td>
</tr>
<tr>
<td>plot_map(map, affine[, threshold])</td>
<td>Plot a 3D map in all the views.</td>
</tr>
<tr>
<td>title(text[, x, y, size, color, bgcolor, alpha])</td>
<td>Write a title to the view.</td>
</tr>
</tbody>
</table>

__init__(cut_coords, axes=None, black_bg=False)
Create 3 linked axes for plotting orthogonal cuts.

Parameters

cut_coords: 3 tuple of ints:
The cut position, in world space.

axes: matplotlib axes object, optional:
The axes that will be subdivided in 3.

black_bg: boolean, optional:
If True, the background of the figure will be put to black. If you wish to save figures with a black background, you will need to pass “facecolor=’k’, edgecolor=’k’” to matplotlib’s savefig.

annotate(left_right=True, positions=True, size=12, **kwargs)
Add annotations to the plot.

Parameters

left_right: boolean, optional:
If left_right is True, annotations indicating which side is left and which side is right are drawn.

positions: boolean, optional:
If positions is True, annotations indicating the positions of the cuts are drawn.

size: integer, optional:
The size of the text used.

kwargs:
Extra keyword arguments are passed to matplotlib’s text function.

contour_map(map, affine, **kwargs)
Contour a 3D map in all the views.
Parameters  

**map**: 3D ndarray : 

The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

**affine**: 4x4 ndarray : 

The affine matrix giving the transformation from voxel indices to world space.

**kwargs** : 

Extra keyword arguments are passed to contour.

**draw_cross** (cut_coords=None, **kwargs) 

Draw a crossbar on the plot to show where the cut is performed.

Parameters  

cut_coords: 3-tuple of floats, optional : 

The position of the cross to draw. If none is passed, the ortho_slicer’s cut coordinates are used.

**kwargs** : 

Extra keyword arguments are passed to axhline

**edge_map** (map, affine, color='r') 

Plot the edges of a 3D map in all the views.

Parameters  

**map**: 3D ndarray : 

The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

**affine**: 4x4 ndarray : 

The affine matrix giving the transformation from voxel indices to world space.

**color**: matplotlib color: string or (r, g, b) value : 

The color used to display the edge map

**classmethod** find_cut_coords (data=None, affine=None, threshold=None, cut_coords=None) 

**classmethod** init_with_figure (data=None, affine=None, threshold=None, cut_coords=None, figure=None, axes=None, black_bg=False, leave_space=False) 

**plot_map** (map, affine, threshold=None, **kwargs) 

Plot a 3D map in all the views.

Parameters  

**map**: 3D ndarray : 

The 3D map to be plotted. If it is a masked array, only the non-masked part will be plotted.

**affine**: 4x4 ndarray : 

The affine matrix giving the transformation from voxel indices to world space.

**threshold** : a number, None, or ‘auto’ 

If None is given, the maps are not thresholded. If a number is given, it is used to threshold the maps: values below the threshold are plotted as transparent.

**kwargs** : 

Extra keyword arguments are passed to imshow.

**title** (text, x=0.01, y=0.99, size=15, color=None, bgcolor=None, alpha=1, **kwargs) 

Write a title to the view.
Parameters
text: string :
The text of the title

x: float, optional :
The horizontal position of the title on the frame in fraction of the frame width.

y: float, optional :
The vertical position of the title on the frame in fraction of the frame height.

size: integer, optional :
The size of the title text.

color: matplotlib color specifier, optional :
The color of the font of the title.

bgcolor: matplotlib color specifier, optional :
The color of the background of the title.

alpha: float, optional :
The alpha value for the background.

kwargs:
Extra keyword arguments are passed to matplotlib’s text function.

117.3 Function

nipy.labs.viz_tools.slicers.demo_ortho_slicer()
A small demo of the OrthoSlicer functionality.
LABS.VIZ_TOOLS.TEST.TEST_ACTIVATION_MAPS

118.1 Module: labs.viz_tools.test.test_activation_maps

118.2 Functions

nipy.labs.viz_tools.test.test_activation_maps.test_anat_cache()
nipy.labs.viz_tools.test.test_activation_maps.test_demo_plot_map()
nipy.labs.viz_tools.test.test_activation_maps.test_plot_anat()
119.1 Module: labs.viz_tools.test.test_cm

Smoke testing the cm module

119.2 Functions

nipy.labs.viz_tools.test.test_cm.test_dim_cmap()
nipy.labs.viz_tools.test.test_cm.test_replace_inside()
LABS.VIZ_TOOLS.TEST.TEST_COORD_TOOLS

120.1 Module: labs.viz_tools.test.test_coord_tools

120.2 Functions

nipy.labs.viz_tools.test.test_coord_tools.test_coord_transform_trivial()
nipy.labs.viz_tools.test.test_coord_tools.test_find_cut_coords()
LABS.VIZ_TOOLS.TEST.TEST_EDGE_DETECT

121.1 Module: labs.viz_tools.test.test_edge_detect

121.2 Functions

nipy.labs.viz_tools.test.test_edge_detect.test_edge_detect()
nipy.labs.viz_tools.test.test_edge_detect.test_fast_abs_percentile()
LABS.VIZ_TOOLS.TEST.TEST_SLICERS

122.1 Module: labs.viz_tools.test.test_slicers

nipy.labs.viz_tools.test.test_slicers.test_demo_ortho_slicer()
CHAPTER
THREE

MODALITIES.FMRI.DESIGN

123.1 Module: modalities.fmri.design

Convenience functions for specifying a design in the GLM.

123.2 Functions

nipy.modalities.fmri.design.event_design(event_spec, t, order=2, hrfs=[glover])
Create a design matrix for a GLM analysis based on an event specification, evaluating it a sequence of time values. Each column in the design matrix will be convolved with each HRF in hrfs.

Parameters event_spec : np.recarray
A recarray having at least a field named ‘time’ signifying the event time, and all other fields will be treated as factors in an ANOVA-type model.

    t : np.ndarray
An array of np.float values at which to evaluate the design. Common examples would be the acquisition times of an fMRI image.

    order : int
The highest order interaction to be considered in constructing the contrast matrices.

    hrfs : seq
A sequence of (symbolic) HRF that will be convolved with each event. If empty, glover is used.

Returns X : np.ndarray
The design matrix with X.shape[0] == t.shape[0]. The number of columns will depend on the other fields of event_spec.

    contrasts : dict
Dictionary of contrasts that is expected to be of interest from the event specification. For each interaction / effect up to a given order will be returned. Also, a contrast is generated for each interaction / effect for each HRF specified in hrfs.

nipy.modalities.fmri.design.fourier_basis(t, freq)
Create a design matrix with columns given by the Fourier basis with a given set of frequencies.

Parameters t : np.ndarray
An array of np.float values at which to evaluate the design. Common examples would be the acquisition times of an fMRI image.

freq : sequence of float

Frequencies for the terms in the Fourier basis.

Returns X : np.ndarray

Examples

```python
t = np.linspace(0,50,101)
drift = fourier_basis(t, np.array([4,6,8]))
drift.shape
(101, 6)
```

nipy.modalities.fmri.design.natural_spline(tvals, knots=None, order=3, intercept=True)

Create a design matrix with columns given by a natural spline of a given order and a specified set of knots.

Parameters
tvals : np.array

Time values

knots : None or sequence, optional

Sequence of float. Default None (same as empty list)

order : int, optional

Order of the spline. Defaults to a cubic (==3)

intercept : bool, optional

If True, include a constant function in the natural spline. Default is False

Returns X : np.ndarray

Examples

```python
tvals = np.linspace(0,50,101)
drift = natural_spline(tvals, knots=[10,20,30,40])
drift.shape
(101, 8)
```

nipy.modalities.fmri.design.stack2designs(old_X, new_X, old_contrasts={}, new_contrasts={})

Add some columns to a design matrix that has contrasts matrices already specified, adding some possibly new contrasts as well.

This basically performs an np.hstack of old_X, new_X and makes sure the contrast matrices are dealt with accordingly.

If two contrasts have the same name, an exception is raised.

Parameters old_X : np.ndarray

A design matrix

new_X : np.ndarray

A second design matrix to be stacked with old_X

old_contrast : dict
Dictionary of contrasts in the old_X column space

new_contrasts : dict
Dictionary of contrasts in the new_X column space

Returns X : np.ndarray
A new design matrix: np.hstack([old_X, new_X])

contrasts : dict
The new contrast matrices reflecting changes to the columns.

def stack_contrasts(contrasts, name, keys)
Create a new F-contrast matrix called 'name' based on a sequence of keys. The contrast is added to contrasts, in-place.

Parameters contrasts : dict
Dictionary of contrast matrices

name : str
Name of new contrast. Should not already be a key of contrasts.

keys : sequence of str
Keys of contrasts that are to be stacked.

Returns None :

def stack_designs(pairs)
Stack a sequence of design / contrast dictionary pairs. Uses multiple calls to stack2designs

Parameters pairs : sequence filled with (np.ndarray, dict) or np.ndarray

Returns X : np.ndarray
new design matrix: np.hstack([old_X, new_X])

contrasts : dict
The new contrast matrices reflecting changes to the columns.
124.1 Module: modalities.fmri.design_matrix

Inheritance diagram for nipy.modalities.fmri.design_matrix:

```
from nipy.modalities.fmri.design_matrix import DesignMatrix
```

124.2 Class

124.3 DesignMatrix

**class** `nipy.modalities.fmri.design_matrix.DesignMatrix(matrix, names, frametimes=None)`

This is a container for a light-weight class for design matrices. This class is only used to make IO and visualization.

**Methods**

```
show(rescale=True, ax=None)
write_csv(path)
```

**show** *(rescale, ax)*
Visualization of a design matrix

**write_csv** *(path)*
Write self.matrix as a csv file with appropriate column names

```
__init__ (matrix, names, frametimes=None)
show (rescale=True, ax=None)
```

**Parameters**

- **rescale**: bool, optional
  - rescale columns magnitude for visualization or not
ax: axis handle, optional:
Handle to axis onto which we will draw design matrix

Returns ax: axis handle:

write_csv(path)
write self.matrix as a csv file with appropriate column names

Parameters path: string, path of the resulting csv file:

Notes
The frametimes are not written

124.4 Functions

nipy.modalities.fmri.design_matrix.dmtx_from_csv(path, frametimes=None)
Return a DesignMatrix instance from a csv file

Parameters path: string, path of the .csv file:

Returns A DesignMatrix instance:

nipy.modalities.fmri.design_matrix.dmtx_light(frametimes, paradigm=None,
hrf_model='canonical',
drift_model='cosine', hfcut=128,
drift_order=1, fir_delays=[0],
add_regs=None, add_reg_names=None,
path=None)

Make a design matrix while avoiding framework

Parameters see make_dmtx, plus:
path: string, optional: a path to write the output:

Returns dmtx array of shape(nreg, nbframes):
the sampled design matrix
names list of strings of len (nreg):
the names of the columns of the design matrix

nipy.modalities.fmri.design_matrix.make_dmtx(frametimes, paradigm=None,
hrf_model='canonical',
drift_model='cosine', hfcut=128,
drift_order=1, fir_delays=[0],
add_regs=None, add_reg_names=None)

Generate a design matrix from the input parameters

Parameters frametimes: array of shape(nbframes), the timing of the scans:
paradigm: Paradigm instance, optional:
description of the experimental paradigm
hrf_model: string, optional:
that specifies the hemodynamic response function it can be 'canonical', 'canonical with derivative' or 'fir'
drift_model: string, optional :
   specifies the desired drift model, to be chosen among ‘polynomial’, ‘cosine’, ‘blank’

hfcut: float, optional :
   cut frequency of the low-pass filter

drift_order: int, optional :
   order of the drift model (in case it is polynomial)

fir_delays: array of shape(nb_onsets) or list, optional, :
   in case of FIR design, yields the array of delays used in the FIR model

add_regs: array of shape(nbframes, naddreg), optional :
   additional user-supplied regressors

add_reg_names: list of (naddreg) regressor names, optional :
   if None, while naddreg>0, these will be termed ‘reg_%i’,i=0..naddreg-1

Returns  DesignMatrix instance :
MODALITIES.FMRI.EXPERIMENTAL_PARADIGM

125.1 Module: modalities.fmri.experimental_paradigm

Inheritance diagram for nipy.modalities.fmri.experimental_paradigm:

125.2 Classes

125.2.1 BlockParadigm

class nipy.modalities.fmri.experimental_paradigm.BlockParadigm(con_id=None, onset=None, duration=None, amplitude=None):

Bases: nipy.modalities.fmri.experimental_paradigm.Paradigm

Class to handle block paradigms

Methods

write_to_csv(csv_file[, session]) Write the paradigm to a csv file

__init__(con_id=None, onset=None, duration=None, amplitude=None)

Parameters
con_id: array of shape (n_events), type = string, optional :
id of the events (name of the experimental condition)
onset: array of shape (n_events), type = float, optional :
onset time (in s.) of the events

amplitude: array of shape (n_events), type = float, optional, :
amplitude of the events (if applicable)

write_to_csv (csv_file, session='0')
Write the paradigm to a csv file

Parameters
csv_file: string, path of the csv file :
session: string, optional, session identifier :

125.2.2 EventRelatedParadigm

class nipy.modalities.fmri.experimental_paradigm.EventRelatedParadigm(con_id=None, onset=None, amplitude=None)

Bases: nipy.modalities.fmri.experimental_paradigm.Paradigm
Class to handle event-related paradigms

Methods

write_to_csv (csv_file[, session]) Write the paradigm to a csv file

__init__ (con_id=None, onset=None, amplitude=None)

Parameters
con_id: array of shape (n_events), type = string, optional :
id of the events (name of the experimental condition)
onset: array of shape (n_events), type = float, optional :
onset time (in s.) of the events
amplitude: array of shape (n_events), type = float, optional, :
amplitude of the events (if applicable)

write_to_csv (csv_file, session='0')
Write the paradigm to a csv file

Parameters
csv_file: string, path of the csv file :
session: string, optional, session identifier :

125.2.3 Paradigm

class nipy.modalities.fmri.experimental_paradigm.Paradigm(con_id=None, onset=None, amplitude=None)

Bases: object
Simple class to handle the experimental paradigm in one session
Methods

write_to_csv(csv_file[, session])  Write the paradigm to a csv file

__init__(con_id=None, onset=None, amplitude=None)

Parameters  con_id: array of shape (n_events), type = string, optional :
    identifier of the events
onset: array of shape (n_events), type = float, optional, :
    onset time (in s.) of the events
amplitude: array of shape (n_events), type = float, optional, :
    amplitude of the events (if applicable)

write_to_csv(csv_file, session='0')
    Write the paradigm to a csv file

Parameters  csv_file: string, path of the csv file :
    session: string, optional, session identifier :

125.3 Function

nipy.modalities.fmri.experimental_paradigm.load_paradigm_from_csv_file(path,
session= None)

Read a (.csv) paradigm file consisting of values yielding (occurrence time, (duration), event ID, modulation) and returns a paradigm instance or a dictionary of paradigm instances

Parameters  path: string, :
    path to a .csv file that describes the paradigm
session: string, optional, session identifier :
    by default the output is a dictionary of session-level dictionaries indexed by session

Returns  paradigm, paradigm instance (if session is provided), or :
    dictionary of paradigm instances otherwise, the resulting session-by-session paradigm

Notes

It is assumed that the csv file contains the following columns: (session id, condition id, onset), plus possibly (duration) and/or (amplitude). If all the durations are 0, the paradigm will be handled as event-related.
126.1 Module: modalities.fmri.fmri

Inheritance diagram for nipy.modalities.fmri.fmri:

```
image.image_list.ImageList -> fmri.fmri.FmriImageList
```

126.2 FmriImageList

class nipy.modalities.fmri.fmri.FmriImageList (images=None, volume_start_times=None, slice_times=None)

Bases: nipy.core.image.image_list.ImageList

Class to implement image list interface for FMRI time series
Allows metadata such as volume and slice times

Methods

```python
from_image(klass, fourdimage[, axis, ...])  # Create an FmriImageList from a 4D Image
get_list_data([axis])  # Return data in ndarray with list dimension at position axis
next()
```

__init__ (images=None, volume_start_times=None, slice_times=None)

An implementation of an fMRI image as in ImageList

Parameters images : iterable

an iterable object whose items are meant to be images; this is checked by asserting that
each has a coordmap attribute and a get_data method. Note that Image objects are
not iterable by default; use the from_image classmethod or iter_axis function to
convert images to image lists - see examples below for the latter.

volume_start_times: None or float or (N,) ndarray :

start time of each frame. It can be specified either as an ndarray with
N=len(images) elements or as a single float, the TR. None results in
np.arange(len(images)).astype(np.float)

slice_times: None or (N,) ndarray :

specifying offset for each slice of each frame, from the frame start time.

See Also:
nipy.core.image_list.ImageList

Examples

```python
>>> from nipy.testing import funcfile
>>> from nipy.io.api import load_image
>>> from nipy.core.api import iter_axis

>>> funcim = load_image(funcfile)
>>> iterable_img = iter_axis(funcim, 't')
>>> fmrilist = FmriImageList(iterable_img)

>>> print fmrilist.get_list_data(axis=0).shape
(20, 17, 21, 3)
>>> print fmrilist[4].shape
(17, 21, 3)
```

classmethod from_image (klass, fourdimage=’t’, volume_start_times=None, slice_times=None) Create an FmriImageList from a 4D Image

Get images by extracting 3d images along the ‘t’ axis.

Parameters fourdimage : Image instance

A 4D Image

volume_start_times: None or float or (N,) ndarray :

start time of each frame. It can be specified either as an ndarray with
N=len(images) elements or as a single float, the TR. None results in
np.arange(len(images)).astype(np.float)

slice_times: None or (N,) ndarray :

specifying offset for each slice of each frame, from the frame start time.

Returns filist : FmriImageList instance

get_list_data (axis=None) Return data in ndarray with list dimension at position axis

Parameters axis : int

axis specifies which axis of the output will take the role of the list dimension. For example, 0 will put the list dimension in the first axis of the result.

Returns data : ndarray

data in image list as array, with data across elements of the list concatenated at dimension axis of the array.
Examples

```python
>>> from nipy.testing import funcfile
>>> from nipy.io.api import load_image

>>> funcim = load_image(funcfile)

>>> ilist = ImageList.from_image(funcim, axis='t')

>>> ilist.get_list_data(axis=0).shape
(20, 17, 21, 3)
```

```python
next ()
nipy.modalities.fmri.fmri.axis0_generator (data, slicers=None)

Takes array-like data, returning slices over axes > 0

This function takes an array-like object data and yields tuples of slicing thing and slices like:

```
[slicer, np.asarray(data)[:,slicer] for slicer in slicer]
```

which in the default (slicers is None) case, boils down to:

```
[i, np.asarray(data)[:,i] for i in range(data.shape[1])]
```

This can be used to get arrays of time series out of an array if the time axis is axis 0.

Parameters

data : array-like

object such that arr = np.asarray(data) returns an array of at least 2 dimensions.

slicers : None or sequence

sequence of objects that can be used to slice into array arr returned from data. If None, default is range(data.shape[1])
```
127.1 Module: modalities.fmri.fmristat.delay

This module defines a class to output estimates of delays and contrasts of delays.


127.2 Contrast

class nipy.modalities.fmri.fmristat.delay.Contrast
    Bases: object
    Empty boggus class to get the docs building.
    __init__()
    x.__init__(...) initializes x; see help(type(x)) for signature
MODALITIES.FMRI.FMRISTAT.HRF

128.1 Module: modalities.fmri.fmristat.hrf

Computation of the canonical HRF used in fMRIstat, both the 2-term spectral approximation and the Taylor series approximation, to a shifted version of the canonical Glover HRF.

128.1.1 References


128.2 Functions

nipy.modalities.fmri.fmristat.hrf.spectral_decomposition(hrf2decompose, time=None, delta=None, ncomp=2)

PCA decomposition of symbolic HRF shifted over time

Perform a PCA expansion of a symbolic HRF, time shifted over the values in delta, returning the first ncomp components.

This smooths out the HRF as compared to using a Taylor series approximation.

Parameters

hrf2decompose : sympy expression
    An expression that can be lambdified as a function of ‘t’. This is the HRF to be expanded in PCA.

time : None or np.ndarray, optional
    None gives default value of np.linspace(-15,50,3251) chosen to match fMRIstat implementation. This corresponds to a time interval of 0.02. Presumed to be equally spaced.

delta : None or np.ndarray, optional
    None results in default value of np.arange(-4.5, 4.6, 0.1) chosen to match fMRIstat implementation.

ncomp : int, optional
    Number of principal components to retain.

Returns

hrf : [sympy expressions]
A sequence length \( n_{comp} \) of symbolic HRFs that are the principal components.

**approx:**

TODO

\[
\text{nipy.modalities.fmri.fmristat.hrf.taylor_approx}(hrf2decompose, \text{time}=\text{None}, \text{delta}=\text{None})
\]

A Taylor series approximation of an HRF shifted by times \( \delta \).

Returns original HRF and gradient of HRF.

**Parameters**

- \( \text{hrf2decompose} \): sympy expression
  - An expression that can be lambdified as a function of `t`. This is the HRF to be expanded in PCA.

- \( \text{time} \): None or np.ndarray, optional
  - None gives default value of np.linspace(-15,50,3251) chosen to match fMRIstat implementation. This corresponds to a time interval of 0.02. Presumed to be equally spaced.

- \( \text{delta} \): None or np.ndarray, optional
  - None results in default value of np.arange(-4.5, 4.6, 0.1) chosen to match fMRIstat implementation.

**Returns**

- \( \text{hrf} \): [sympy expressions]
  - Sequence length 2 comprising \((hrf2decompose, dhrf)\) where \( dhrf \) is the first derivative of \( hrf2decompose \).

**approx:**

TODO

**References**

129.1 Module: modalities.fmri.fmristat.invert

nipy.modalities.fmri.fmristat.invert\_invertR(delta, IRF, niter=20)

If IRF has 2 components (w0, w1) return an estimate of the inverse of r=w1/w0, as in Liao et al. (2002). Fits a simple arctan model to the ratio w1/w0.
This module defines the two default GLM passes of fmristat

The results of both passes of the GLM get pushed around by generators, which know how to get out the (probably 3D) data for each slice, or parcel (for the AR) case, estimate in 2D, then store the data back again in its original shape.

The containers here, in the execute methods, know how to reshape the data on the way into the estimation (to 2D), then back again, to 3D, or 4D.

It’s relatively easy to do this when just iterating over simple slices, but it gets a bit more complicated when taking arbitrary shaped samples from the image, as we do for estimating the AR coefficients, where we take all the voxels with similar AR coefficients at once.

130.2 Classes

130.2.1 AR1

class nipy.modalities.fmri.fmristat.model.AR1(fmri_image, formula, rho, outputs=[], volume_start_times=None) Bases: object

Second pass through fmri_image.

Parameters

fmri_image : FmriImageList
object returning 4D array from np.asarray, having attribute
volume_start_times (if volume_start_times is None), and such that object[0]
returns something with attributes shape

formula : nipy.algorithms.statistics.formula.Formula
rho : Image
    image of AR(1) coefficients. Returning data from rho.get_data(), and having
attribute coordmap
outputs :
volume_start_times :

Methods

execute()

__init__ (fmri_image, formula, rho, outputs=[], volume_start_times=None)
execute()

130.2.2 ModelOutputImage
class nipy.modalities.fmri.fmristat.model.ModelOutputImage (filename, coordmap,
shape, clobber=False)
Bases: object

These images have their values filled in as the model is fit, and are saved to disk after being completely filled in.
They are saved to disk by calling the ‘save’ method.
The __getitem__ and __setitem__ calls are delegated to a private Image. An exception is raised if trying to
get/set data after the data has been saved to disk.

Methods

save()

__init__ (filename, coordmap, shape, clobber=False)
save()
    Save current Image data to disk

130.2.3 OLS
class nipy.modalities.fmri.fmristat.model.OLS (fmri_image, formula, outputs=[], volume_start_times=None)
Bases: object

First pass through fmri_image.

Parameters   fmri_image : FmriImageList or 4D image
object returning 4D data from np.asarray, with first (object[0]) axis being the independent variable of the model; object[0] returns an object with attribute shape.

    formula : nipy.algorithms.statistics.formula.Formula
    outputs : :
    volume_start_times :

Methods

    execute()

    __init__(fmri_image, formula, outputs=[], volume_start_times=None)
    execute()

130.3 Functions

nipy.modalities.fmri.fmristat.model.estimateAR(resid, design, order=1)
Estimate AR parameters using bias correction from fMRIstat.

    Parameters resid : array-like :
    residuals from model
    model : an OLS model used to estimate residuals :
    Returns output : array
    shape (order, resid
nipy.modalities.fmri.fmristat.model.generate_output(outputs, iterable, reshape=<function reshape=<function lambda> at 0x51b3c80>)
Write out results of a given output.
In the regression setting, results is generally going to be a scipy.stats.models.model.LikelihoodModelResults instance.

    Parameters outputs : sequence
    sequence of output objects
    iterable : object
    Object which iterates, returning tuples of (indexer, results), where indexer can be used to index into the outputs
    reshape : callable
    accepts two arguments, first is the indexer, and the second is the array which will be indexed; returns modified indexer and array ready for slicing with modified indexer.
nipy.modalities.fmri.fmristat.model.model_generator(formula, data, volume_start_times, iterable=None, slicetimes=None, model_type=<class 'nipy.algorithms.statistics.models.regression.OLSModel'>, model_params=<function <lambda> at 0x51b3500>)

Generator for the models for a pass of fmristat analysis.

nipy.modalities.fmri.fmristat.model.output_AR1(outfile, fmri_image, clobber=False)

Create an output file of the AR1 parameter from the OLS pass of fmristat.

Parameters

outfile :
  
  fmri_image : FmriImageList or 4D image
    object such that object[0] has attributes coordmap and shape
  clobber : bool
    if True, overwrite previous output

Returns
  regression_output : RegressionOutput instance

nipy.modalities.fmri.fmristat.model.output_F(outfile, contrast, fmri_image, clobber=False)

output F statistic images

Parameters

outfile : str
  filename for F contrast image

contrast : array
  F contrast matrix

fmri_image : FmriImageList or Image
  object such that object[0] has attributes shape and coordmap

Returns
  f_reg_out : RegressionOutput instance

Object that can a) be called with a results instance as argument, returning an array, and
b) accept the output array for storing, via obj[slice_spec] = arr type slicing.

nipy.modalities.fmri.fmristat.model.output_T(outbase, contrast, fmri_image, effect=True, sd=True, t=True, clobber=False)

Return t contrast regression outputs list for contrast

Parameters

outbase : string
  Base filename that will be used to construct a set of files for the TContrast. For example, outbase='output.nii' will result in the following files (assuming defaults for all other params): output_effect.nii, output_sd.nii, output_t.nii

contrast : array
  F contrast matrix

fmri_image : FmriImageList or Image
  object such that object[0] has attributes shape and coordmap

Returns
  t_reg_out : RegressionOutput instance

Object that can a) be called with a results instance as argument, returning an array, and
b) accept the output array for storing, via obj[slice_spec] = arr type slicing.
whether to write an effect image

sd : {True, False}, optional
whether to write a standard deviation image

t : {True, False}, optional
whether to write a t image

clobber : {False, True}, optional
whether to overwrite images that exist.

Returns reglist: RegressionOutputList instance

Regression output list with selected outputs, where selection is by inputs effect, sd and t

Notes

Note that this routine uses the corresponding output_T routine in outputters, but indirectly via the TOutput object.

nipy.modalities.fmri.fmristat.model.output_resid(outfile, fmri_image, clobber=False)
Create an output file of the residuals parameter from the OLS pass of fmristat.

Uses affine part of the first image to output resid's unless fmri_image is an Image.

Parameters outfile :

fmri_image : FmriImageList or 4D image

If FmriImageList, needs attributes volume_start_times, supports len(), and object[0] has attributes affine, coordmap and shape, from which we create a new 4D coordmap and shape. If 4D image, use the images coordmap and shape.

clobber : bool
if True, overwrite previous output

Returns regression_output :

nipy.modalities.fmri.fmristat.model.results_generator(model_iterable)
Generator for results from an iterator that returns (index, data, model) tuples.

See model_generator.
MODALITIES.FMRI.FMRISTAT.OUTPUTTERS

131.1 Module: modalities.fmrifmristat.outputters

Convenience functions and classes for statistics on images.
These functions and classes support the return of statistical test results from iterations through data.
The basic container here is the RegressionOutput. This does two basic things:
• via __call__, processes a result object from a regression to produce something, usually an array
• via slicing (__setitem__), it can store stuff, usually arrays.
We use these by other objects (see algorithms.statistics.fmrifmristat) slicing data out of images, fitting models to the
data to create results objects, and then passing them to these here RegressionOutput containers via call, to get
useful arrays, and then putting the results back into the RegressionOutput containers via slicing (__setitem__).

131.2 Classes

131.2.1 RegressionOutput
class nipy.modalities.fmrifmristat.outputters.RegressionOutput(img, fn, output_shape=None)
    Bases: object
    A class to output things in GLM passes through arrays of data.
Methods

__call__(x)

__init__(img, fn, output_shape=None)
  Parameters  img : Image instance
                  The output Image
  fn : callable
       A function that is applied to a models.model.LikelihoodModelResults instance

131.2.2 RegressionOutputList

class nipy.modalities.fmri.fmristat.outputters.RegressionOutputList (imgs, fn)
  Bases: object

  A class to output more than one thing from a GLM pass through arrays of data.

Methods

__call__(x)

__init__(imgs, fn)
  Initialize regression output list
  Parameters  imgs : list
                  The list of output images
  fn : callable
       A function that is applied to a models.model.LikelihoodModelResults instance

131.2.3 TOutput

class nipy.modalities.fmri.fmristat.outputters.TOutput (contrast, effect=None, sd=None, t=None)
  Bases: nipy.modalities.fmri.fmristat.outputters.RegressionOutputList

  Output contrast related to a T contrast from a GLM pass through data.

Methods

__call__(x)

__init__(contrast, effect=None, sd=None, t=None)
131.3 Functions

`nipy.modalities.fmri.fmristat.outputters.output_AR1(results)`

Compute the usual AR(1) parameter on the residuals from a regression.

`nipy.modalities.fmri.fmristat.outputters.output_F(results, contrast)`

This convenience function outputs the results of an Fcontrast from a regression.

- **Parameters**
  - `results`: object implementing Tcontrast method
  - `contrast`: array
    - contrast matrix

- **Returns**
  - `F`: array
    - array of F values

`nipy.modalities.fmri.fmristat.outputters.output_T(results, contrast, retvals=('effect', 'sd', 't'))`

Convenience function to collect t contrast results.

- **Parameters**
  - `results`: object implementing Tcontrast method
  - `contrast`: array
    - contrast matrix
  - `retvals`: sequence, optional
    - None or more of strings ‘effect’, ‘sd’, ‘t’, where the presence of the string means that that output will be returned.

- **Returns**
  - `res_list`: list
    - List of results. It will have the same length as `retvals` and the elements will be in the same order as `retvals`

`nipy.modalities.fmri.fmristat.outputters.output_resid(results)`

This convenience function outputs the residuals from a regression.
132.1 Module: modalities.fmri.glm

Inheritance diagram for nipy.modalities.fmri.glm:

This module presents an interface to use the glm implemented in nipy.algorithms.statistics.models.regression. It contains the GLM and contrast classes that are meant to be the main objects of fMRI data analyses. It is important to note that the GLM is meant as a one-session General Linear Model. But inference can be performed on multiple sessions by computing fixed effects on contrasts

132.1.1 Examples

```python
>>> import numpy as np
>>> from nipy.modalities.fmri.glm import GeneralLinearModel
>>> n, p, q = 100, 80, 10
>>> X, Y = np.random.randn(p, q), np.random.randn(p, n)
>>> cval = np.hstack((1, np.zeros(9)))
>>> model = GeneralLinearModel(X)
>>> model.fit(Y)
>>> z_vals = model.contrast(cval).z_score()  # z-transformed statistics
```

Example of fixed effects statistics across two contrasts
>>> cval_ = cval.copy()
>>> np.random.shuffle(cval_)
>>> z_ffx = (model.contrast(cval) + model.contrast(cval_)).z_score()

## 132.2 Classes

### 132.2.1 Contrast

class `nipy.modalities.fmri.glm.Contrast`:

```python
class nipy.modalities.fmri.glm.Contrast(effect, variance, dof=10000000000.0, contrast_type='t', tiny=1e-50, dofmax=10000000000.0)
```

The contrast class handles the estimation of statistical contrasts on a given model: student (t), Fisher (F), conjunction (tmin-conjunction). The important feature is that it supports addition, thus opening the possibility of fixed-effects models.

The current implementation is meant to be simple, and could be enhanced in the future on the computational side (high-dimensional F constraints may lead to memory breakage).

### Notes

The ‘tmin-conjunction’ test is the valid conjunction test discussed in: Nichols T, Brett M, Andersson J, Wager T, Poline JB. Valid conjunction inference with the minimum statistic. Neuroimage. 2005 Apr 15;25(3):653-60. This test gives the p-value of the z-values under the conjunction null, i.e. the union of the null hypotheses for all terms.

### Methods

- `p_value([baseline])` Return a parametric estimate of the p-value associated with the null hypothesis: (H0) ‘contrast equals baseline’
- `stat([baseline])` Return the decision statistic associated with the test of the null hypothesis.
- `z_score([baseline])` Return a parametric estimation of the z-score associated with the null hypothesis.

```python
__init__(effect, variance, dof=10000000000.0, contrast_type='t', tiny=1e-50, dofmax=10000000000.0)
```

Parameters:
- `effect`: array of shape (contrast_dim, n_voxels):
  - the effects related to the contrast
- `variance`: array of shape (contrast_dim, contrast_dim, n_voxels):
  - the associated variance estimate
- `dof`: scalar, the degrees of freedom:
- `contrast_type`: string to be chosen among ‘t’ and ‘F’:

```python
p_value(baseline=0.0)
```

Return a parametric estimate of the p-value associated with the null hypothesis: (H0) ‘contrast equals baseline’

Parameters:
- `baseline`: float, optional:
  - Baseline value for the test statistic
\texttt{stat}(\texttt{baseline}=0.0) \\
Return the decision statistic associated with the test of the null hypothesis: (H0) ‘contrast equals baseline’

\textbf{Parameters} \hspace{1em} \texttt{baseline}: float, optional, :
Baseline value for the test statistic

\texttt{z\_score}(\texttt{baseline}=0.0) \\
Return a parametric estimation of the z-score associated with the null hypothesis: (H0) ‘contrast equals baseline’

\textbf{Parameters} \hspace{1em} \texttt{baseline}: float, optional, :
Baseline value for the test statistic

132.2.2 \texttt{FMRILinearModel}

class \texttt{nipy.modalities.fmri.glm.FMRILinearModel}(\texttt{fmri\_data}, \texttt{design\_matrices}, mask='compute', m=0.2, M=0.9, threshold=0.5) \\
\textbf{Bases}: object
This class is meant to handle GLMs from a higher-level perspective i.e. by taking images as input and output

\textbf{Methods}

\texttt{contrast}(\texttt{contrasts[, con\_id, contrast\_type, ...]}) \hspace{1em} \text{Estimation of a contrast as fixed effects on all sessions}

\texttt{fit}([\texttt{do\_scaling, model, steps}]) \hspace{1em} \text{Load the data, mask the data, scale the data, fit the GLM}

\texttt{\_\_init\_\_}(\texttt{fmri\_data, design\_matrices, mask=’compute’, m=0.2, M=0.9, threshold=0.5}) \hspace{1em} \text{Load the data}

\textbf{Parameters} \hspace{1em} \texttt{fmri\_data}: Image or str or sequence of Images / str
fmri images / paths of the (4D) fmri images
\texttt{design\_matrices}: arrays or str or sequence of arrays / str
design matrix arrays / paths of .npz files
\texttt{mask}: str or Image or None, optional
string can be ‘compute’ or a path to an image image is an input (assumed binary) mask image(s), if ‘compute’, the mask is computed if None, no masking will be applied
\texttt{m, M, threshold}: float, optional:
parameters of the masking procedure. Should be within [0, 1]

\textbf{Notes}

The only computation done here is mask computation (if required)

\textbf{Examples}

We need the example data package for this example
```
>>> from nipy.utils import example_data
>>> from nipy.modalities.fmri.glm import FMRILinearModel

>>> fmri_files = [example_data.get_filename('fiac', 'fiac0', run)
... for run in ['run1.nii.gz', 'run2.nii.gz']]

>>> design_files = [example_data.get_filename('fiac', 'fiac0', run)
... for run in ['run1_design.npz', 'run2_design.npz']]

>>> mask = example_data.get_filename('fiac', 'fiac0', 'mask.nii.gz')

>>> multi_session_model = FMRILinearModel(fmri_files, design_files, mask)

>>> multi_session_model.fit()

>>> z_image, = multi_session_model.contrast([np.eye(13)[1]] * 2)

The number of voxels with p < 0.001

>>> np.sum(z_image.get_data() > 3.09)

671

contrast (contrasts, con_id='', contrast_type=None, output_z=True, output_stat=False, output_effects=False, output_variance=False)

Estimation of a contrast as fixed effects on all sessions

Parameters

contrasts : array or list of arrays of shape (n_col) or (n_dim, n_col)

where n_col is the number of columns of the design matrix, numerical definition of
the contrast (one array per run)

con_id : str, optional

name of the contrast

contrast_type : {'t', 'F', 'tmin-conjunction'}, optional

type of the contrast

output_z : bool, optional

Return or not the corresponding z-stat image

output_stat : bool, optional

Return or not the base (t/F) stat image

output_effects : bool, optional

Return or not the corresponding effect image

output_variance : bool, optional

Return or not the corresponding variance image

Returns

output_images : list of nibabel images

The desired output images

fit (do_scaling=True, model='ar1', steps=100)

Load the data, mask the data, scale the data, fit the GLM

Parameters

do_scaling : bool, optional

if True, the data should be scaled as pourcent of voxel mean

model : string, optional,

the kind of glm (‘ols’ or ‘ar1’) you want to fit to the data

steps : int, optional

in case of an ar1, discretization of the ar1 parameter
```
132.2.3 GeneralLinearModel

class nipy.modalities.fmri.glm.GeneralLinearModel(X)
Bases: object

This class handles the so-called on General Linear Model

Most of what it does in the fit() and contrast() methods fit() performs the standard two-step (‘ols’ then ‘ar1’)
GLM fitting contrast() returns a contrast instance, yielding statistics and p-values. The link between fit() and
contrast is done via the two class members:

glm_results [dictionary of nipy.algorithms.statistics.models.] regression.RegressionResults instances, describing
results of a GLM fit

labels [array of shape(n_voxels),] labels that associate each voxel with a results key

Methods

contrast(con_val[, contrast_type]) Specify and estimate a linear contrast
fit(Y[, model, steps]) GLM fitting of a dataset using ‘ols’ regression or the two-pass
get_beta([column_index]) Acceptor for the best linear unbiased estimated of model parameters
get_logL() Acceptor for the log-likelihood of the model
get_mse() Acceptor for the mean squared error of the model

__init__ (X)

Parameters X: array of shape (n_time_points, n_regressors)
the design matrix

contrast (con_val, contrast_type=None)
Specify and estimate a linear contrast

Parameters con_val: numpy.ndarray of shape (p) or (q, p)

where q = number of contrast vectors and p = number of regressors

contrast_type: {None, ‘t’, ‘F’ or ‘tmin-conjunction’}, optional

type of the contrast. If None, then defaults to ‘t’ for 1D con_val and ‘F’ for 2D con_val

Returns con: Contrast instance:

fit(Y, model=’ar1’, steps=100) GLM fitting of a dataset using ‘ols’ regression or the two-pass

Parameters Y: array of shape(n_time_points, n_samples)
the fMRI data

model: {‘ar1’, ‘ols’}, optional

the temporal variance model. Defaults to ‘ar1’

steps: int, optional

Maximum number of discrete steps for the AR(1) coef histogram

get_beta(column_index=None) Acceptor for the best linear unbiased estimated of model parameters

Parameters column_index: int or array-like of int or None, optional:
The indexed of the columns to be returned. if None (default behaviour), the whole vector is returned

Returns  beta: array of shape (n_voxels, n_columns) :
          the beta

get_logL()
Acessor for the log-likelihood of the model

Returns  logL: array of shape (n_voxels,) :
          the sum of square error per voxel

get_mse()
Acessor for the mean squared error of the model

Returns  mse: array of shape (n_voxels) :
          the sum of square error per voxel

132.3 Function

nipy.modalities.fmri.glm.data_scaling(Y)
Scaling of the data to have pourcent of baseline change columnwise

Parameters  Y: array of shape(n_time_points, n_voxels) :
          the input data

Returns  Y: array of shape (n_time_points, n_voxels), :
          the data after mean-scaling, de-meaning and multiplication by 100

mean : array of shape (n_voxels,)
          the data mean
MODALITIES.FMRI.HEMODYNAMIC_MODELS

133.1 Module: modalities.fmri.hemodynamic_models

This module is for canonical hrf specification. Here we provide for SPM, Glover hrf and finite impulse response (FIR) models. This module closely follows SPM implementation

Author: Bertrand Thirion, 2011

133.2 Functions

nipy.modalities.fmri.hemodynamic_models.compute_regressor(exp_condition, hrf_model, frametimes, con_id='cond', oversampling=16, fir_delays=None)

This is the main function to convolve regressors with hrf model

Parameters
exp_condition: descriptor of an experimental condition:

hrf_model: string, the hrf model to be used. Can be chosen among:

'spm', 'spm_time', 'spm_time_dispersion', 'canonical', 'canonical_derivative', 'fir'

frametimes: array of shape (n): the sought:

con_id: string, optional identifier of the condition:

oversampling: int, optional, oversampling factor to perform the convolution:

fir_delays: array-like of int, onsets corresponding to the fir basis:

Returns creg: array of shape(n_scans, n_reg): computed regressors sampled:

at frametimes

reg_names: list of strings, corresponding regressor names:

Notes

The different hemodynamic models can be understood as follows: 'spm': this is the hrf model used in spm
'spm_time': this is the spm model plus its time derivative (2 regressors) 'spm_time_dispersion': idem, plus dispersion derivative (3 regressors) 'canonical': this one corresponds to the Glover hrf 'canonical_derivative':
the Glover hrf + time derivative (2 regressors) 'fir': finite impulse response basis, a set of delayed dirac models
with arbitrary length. This one currently assumes regularly spaced frametimes (i.e. fixed time of repetition).

It is expected that spm standard and Glover model would not yield large differences in most cases.

```python
nipy.modalities.fmri.hemodynamic_models.gamma_difference_hrf(tr, oversampling=16, time_length=32.0, onset=0.0, delay=6, undershoot=16.0, dispersion=1.0, u Dispersion=1.0, ratio=0.167)
```

Compute an hrf as the difference of two gamma functions

**Parameters**
- `tr`: float, scan repeat time, in seconds
- `oversampling`: int, temporal oversampling factor, optional
- `time_length`: float, hrf kernel length, in seconds
- `onset`: float, onset of the hrf

**Returns**
- `hrf`: array of shape(length / tr * oversampling, float), hrf sampling on the oversampled time grid

```python
nipy.modalities.fmri.hemodynamic_models.glover_hrf(tr, oversampling=16, time_length=32.0, onset=0.0)
```

Implementation of the Glover hrf model

**Parameters**
- `tr`: float, scan repeat time, in seconds
- `oversampling`: int, temporal oversampling factor, optional
- `time_length`: float, hrf kernel length, in seconds
- `onset`: float, onset of the response

**Returns**
- `hrf`: array of shape(length / tr * oversampling, float), hrf sampling on the oversampled time grid

```python
nipy.modalities.fmri.hemodynamic_models.glover_time_derivative(tr, oversampling=16, time_length=32.0, onset=0.0)
```

Implementation of the flover time derivative hrf (dhrf) model

**Parameters**
- `tr`: float, scan repeat time, in seconds
- `oversampling`: int, temporal oversampling factor, optional
- `time_length`: float, hrf kernel length, in seconds
- `onset`: float, onset of the response

**Returns**
- `dhrf`: array of shape(length / tr, float), dhrf sampling on the provided grid

```python
nipy.modalities.fmri.hemodynamic_models.resampleRegressor(hr_regressor, hr_frametimes, frametimes, kind='linear')
```

this function samples the regressors at frametimes
Parameters  hr_regressor: array of shape(n), :
            the regressor time course sampled at high temporal resolution

hr_frametimes: array of shape(n), :
            the corresponding time stamps

frametimes: array of shape(p), :
            the desired time stamps

kind: string, optional, the kind of desired interpolation :

Returns  regressor: array of shape(p), the resampled regressor :

    nipy.modalities.fmri.hemodynamic_models.sample_condition(exp_condition, frametimes, over sampling=16)

this function samples the experimental condition at frametimes

Parameters  exp_condition: a tuple of 3 arrays of shape n, corresponding :
            to (onsets, duration, value), describing the experimental condition

frametimes: array of shape(n) :

over_sampling: int, over_sampling factor :

Returns  regressor: array of shape(n) :

    nipy.modalities.fmri.hemodynamic_models.spm_dispersion_derivative(tr, over sampling=16, time_length=32.0, onset=0.0)

Implementation of the SPM dispersion derivative hrf model

Parameters  tr: float, scan repeat time, in seconds :
               oversampling: int, temporal oversampling factor, optional :
               time_length: float, hrf kernel length, in seconds :
               onset: float, onset of the response :

Returns  dhrf: array of shape(length / tr * oversampling, float), :
               dhrf sampling on the oversampled time grid

    nipy.modalities.fmri.hemodynamic_models.spm_hrf(tr, oversampling=16, time_length=32.0, onset=0.0)

Implementation of the SPM hrf model

Parameters  tr: float, scan repeat time, in seconds :
               oversampling: int, temporal oversampling factor, optional :
               time_length: float, hrf kernel length, in seconds :
               onset: float, onset of the response :

Returns  hrf: array of shape(length / tr * oversampling, float), :
               hrf sampling on the oversampled time grid

    nipy.modalities.fmri.hemodynamic_models.spm_time_derivative(tr, oversampling=16, time_length=32.0, onset=0.0)

Implementation of the SPM time derivative hrf (dhrf) model
Parameters  
- tr: float, scan repeat time, in seconds:
  - oversampling: int, temporal oversampling factor, optional:
  - time_length: float, hrf kernel length, in seconds:
  - onset: float, onset of the response:

Returns  
- dhrf: array of shape(length / tr, float):
  - dhrf sampling on the provided grid
134.1 Module: modalities.fmri.hrf

This module provides definitions of various hemodynamic response functions (hrf).
In particular, it provides Gary Glover’s canonical HRF, AFNI’s default HRF, and a spectral HRF.
The Glover HRF is based on:

}

This parameterization is from fmristat:
http://www.math.mcgill.ca/keith/fmristat/

fmristat models the HRF as the difference of two gamma functions, g1 and g2, each defined by the timing of the gamma function peaks (pk1, pk2) and the fwhms (width1, width2):

raw_hrf = g1(pk1, width1) - a2 * g2(pk2, width2)

where a2 is the scale factor for the g2 gamma function. The actual hrf is the raw hrf set to have an integral of 1.

fmristat used pk1, width1, pk2, width2, a2 = (5.4, 5.2, 10.8, 7.35, 0.35). These are parameters to match Glover’s 1 second duration auditory stimulus curves. Glover wrote these as:

y(t) = c1 * t**n1 * exp(t/t1) - a2 * c2 * t**n2 * exp(t/t2)

with n1, t1, n2, t2, a2 = (6.0, 0.9, 12, 0.9, 0.35). The difference between Glover’s expression and ours is because we (and fmristat) use the peak location and width to characterize the function rather than n1, t1. The values we use are equivalent. Specifically, in our formulation:

```python
>>> n1, t1, c1 = gamma_params(5.4, 5.2)
>>> np.allclose((n1-1, t1), (6.0, 0.9), rtol=0.02)
True
>>> n2, t2, c2 = gamma_params(10.8, 7.35)
>>> np.allclose((n2-1, t2), (12.0, 0.9), rtol=0.02)
True
```

134.2 Functions

nipy.modalities.fmri.hrf.gamma_expr(peak_location, peak_fwhm)
Parameters for gamma density given peak and width

TODO: where does the coef come from again... check fmristat code

From a peak location and peak fwhm, determine the parameters (shape, scale) of a Gamma density:

\[ f(x) = \text{coef} \times x^{\text{shape}-1} \times \exp(-x/\text{scale}) \]

The coefficient returned ensures that the \( f \) has integral 1 over \([0, \infty]\)

- **Parameters**: `peak_location` : float
  Location of the peak of the Gamma density

  - `peak_fwhm` : float
    FWHM at the peak

  - **Returns**: `shape` : float
    Shape parameter in the Gamma density

    - `scale` : float
      Scale parameter in the Gamma density

    - `coef` : float
      Coefficient needed to ensure the density has integral 1.
135.1 Module: modalities.fmri.spm.correlation

135.2 Functions

nipy.modalities.fmri.spm.correlation.ARcomponents(rho, n, drho=0.05, cor=False, sigma=1)

Numerically differentiate covariance matrices of AR(p) of length n with respect to AR parameters around the value rho.

If drho is a vector, they are treated as steps in the numerical differentiation.

nipy.modalities.fmri.spm.correlation.ARcovariance(rho, n, cor=False, sigma=1.0)

Return covariance matrix of a sample of length n from an AR(p) process with parameters rho.

INPUTS:

rho – an array of length p 
sigma – standard deviation of the white noise
136.1 Module: modalities.fmri.spm.model

Inheritance diagram for nipy.modalities.fmri.spm.model:

```
spm.model.SecondStage
```

136.2 Class

136.3 SecondStage

```
class nipy.modalities.fmri.spm.model.SecondStage(fmri_image, formula, sigma, outputs=[], volume_start_times=None):

Bases: object

Parameters

fmri_image : FmriImageList
    object returning 4D array from np.asarray, having attribute
    volume_start_times (if volume_start_times is None), and such that object[0]
    returns something with attributes shape

formula : nipy.algorithms.statistics.formula.Formula

sigma :

outputs :

volume_start_times :
```

Methods
execute()

```python
__init__(fmri_image, formula, sigma, outputs=[], volume_start_times=None)
execute()
```

136.4 Functions

```
nipy.modalities.fmri.spm.model.Fmask(Fimg, dfnum, dfdenom, pvalue=0.0001)
Create mask for use in estimating pooled covariance based on an F contrast.

nipy.modalities.fmri.spm.model.estimate_pooled_covariance(resid, ARtarget=[0.3], mask=None)
Use SPM’s REML implementation to estimate a pooled covariance matrix.
Thresholds an F statistic at a marginal pvalue to estimate covariance matrix.
```
137.1 Module: modalities.fmri.spm.reml

137.2 Functions

nipy.modalities.fmri.spm.reml.orth(X, tol=1e-07)
Compute orthonormal basis for the column span of X.

Inputs: X – n-by-p matrix

Outputs:

B – n-by-rank(X) matrix with orthonormal columns spanning the column rank of X

nipy.modalities.fmri.spm.reml.reml(sigma, components, design=None, n=1, niter=128, penalty_cov=1.2664165549094176e-14, penalty_mean=0)
Adapted from spm_reml.m
ReML estimation of covariance components from sigma using design matrix.

Inputs: sigma – m-by-m covariance matrix components – q-by-m-by-m array of variance components mean of sigma is modeled as a some over components[i]

design – m-by-p design matrix whose effect is to be removed for ReML. If None, no effect removed (???)
n – degrees of freedom of sigma penalty_cov – quadratic penalty to be applied in Fisher algorithm.
    If the value is a float, f, the penalty is f * identity(m). If the value is a 1d array, this is the diagonal of the penalty.

penalty_mean – mean of quadratic penalty to be applied in Fisher algorithm. If the value is a float, f, the location is f * np.ones(m).

Outputs: C – estimated mean of sigma h – array of length q representing coefficients of variance components
cov_h – estimated covariance matrix of h
138.1 Module: modalities.fmri.spm.trace

nipy.modalities.fmri.spm.trace.trRV(X=None, V=None)

If V is None it defaults to identity.
If X is None, it defaults to the 0-dimensional subspace, i.e. R is the identity.

```python
>>> import numpy as np
>>> from numpy.random import standard_normal

>>> X = standard_normal((100, 4))
>>> np.allclose(trRV(X), (96.0, 96.0))
True

>>> V = np.identity(100)
>>> np.allclose(trRV(X), (96.0, 96.0))
True

>>> X[:,3] = X[:,1] + X[:,2]
>>> np.allclose(trRV(X), (97.0, 97.0))
True

>>> u = orth(X)
>>> V = np.dot(u, u.T)
>>> print np.allclose(trRV(X, V), 0)
True
```
139.1 Module: modalities.fmri.utils

This module defines some convenience functions of time.

- interp: an expression for an interpolated function of time
  - linear_interp: an expression for a linearly interpolated function of time
- step_function: an expression for a step function of time
- events: a convenience function to generate sums of events
- blocks: a convenience function to generate sums of blocks
- convolve_functions: numerically convolve two functions of time
- fourier_basis: a convenience function to generate a Fourier basis

139.2 Functions

```python
nipy.modalities.fmri.utils.blocks(intervals, amplitudes=None, name=None)
```

Step function based on a sequence of intervals.

**Parameters**

- `intervals`: (S,) sequence of (2,) sequences
  - Sequence (S0, S1, ... S(N-1)) of sequences, where S0 (etc) are sequences of length 2, giving 'on' and 'off' times of block
- `amplitudes`: (S,) sequence of float, optional
  - Optional amplitudes for each block. Defaults to 1.
- `name`: None or str, optional
  - Name of the convolved function in the resulting expression. Defaults to one created by `utils.interp`.

**Returns**

- `b_of_t`: sympy expr
  - Sympy expression b(t) where b is a sympy anonymous function of time that implements the block step function
Examples

```python
>>> on_off = [[1,2],[3,4]]
>>> tval = np.array([0.4,1.4,2.4,3.4])
>>> b = blocks(on_off)
>>> lam = lambdify_t(b)
>>> lam(tval)
array([ 0., 1., 0., 1.])
>>> b = blocks(on_off, amplitudes=[3,5])
>>> lam = lambdify_t(b)
>>> lam(tval)
array([ 0., 3., 0., 5.])
```

nipy.modalities.fmri.utils.\texttt{convolve\_functions}(f, g, f\_interval, g\_interval, dt, fill=0, name=None, **kwargs)

Expression containing numerical convolution of $f_n1$ with $f_n2$

**Parameters**

- \texttt{f} : \texttt{sympy expr}
  An expression that is a function of t only.

- \texttt{g} : \texttt{sympy expr}
  An expression that is a function of t only.

- \texttt{f\_interval} : (2,) sequence of float
  The start and end of the interval of t over which to convolve values of f

- \texttt{g\_interval} : (2,) sequence of floats
  Start and end of the interval of t over to convolve g

- \texttt{dt} : float
  Time step for discretization. We use this for creating the interpolator to form the numerical implementation

- \texttt{fill} : None or float
  Value to return from sampling output $fg$ function outside range.

- \texttt{name} : None or str, optional
  Name of the convolved function in the resulting expression. Defaults to one created by \texttt{utils.interp}.

- **\texttt{kwargs}** : keyword args, optional
  Any other arguments to pass to the \texttt{interp1d} function in creating the numerical function for $fg$.

**Returns**

- \texttt{fg} : \texttt{sympy expr}
  An symbolic expression that is a function of t only, and that can be lambdified to produce a function returning the convolved series from an input array.

Examples

```python
>>> import sympy
>>> t = sympy.Symbol('t')

This is a square wave on [0,1]
```
>>> f1 = (t > 0) * (t < 1)

The convolution of \( f1 \) with itself is a triangular wave on \([0, 2]\), peaking at 1 with height 1

>>> tri = convolve_functions(f1, f1, [0, 2], [0, 2], 1.0e-3, name='conv')

The result is a symbolic function

>>> print tri
cov(t)

Get the numerical values for a time vector

>>> ftri = lambdify(t, tri)

>>> x = np.arange(0, 2, 0.2)

>>> y = ftri(x)

The peak is at 1 >>> x[np.argmax(y)] 1.0

nipy.modalities.fmri.utils.define(name, expr)

Create function of t expression from arbitrary expression expr

Take an arbitrarily complicated expression expr of 't' and make it an expression that is a simple function of t, of form 'f(t)' such that when it evaluates (via lambdify) it has the right values.

Parameters  expr : sympy expression

    with only 't' as a Symbol

    name : str

Returns  nexpr: sympy expression:

Examples

>>> t = Term('t')

>>> expr = t**2 + 3*t

>>> print expr
3*t + t**2

>>> newexpr = define('f', expr)

>>> print newexpr
f(t)

>>> f = lambdify_t(newexpr)

>>> f(4)
28

nipy.modalities.fmri.utils.events(times, amplitudes=None, f=DiracDelta, g=a)

Return a sum of functions based on a sequence of times.

Parameters  times : sequence

    vector of onsets length \(N\)

    amplitudes : None or sequence length \(N\), optional

    Optional sequence of amplitudes. None (default) results in sequence length \(N\) of 1s

    f : sympy.Function, optional
Optional function. Defaults to DiracDelta, can be replaced with another function, f, in which case the result is the convolution with f.

\[ g : \text{sympy.Basic, optional} \]

Optional sympy expression function of amplitudes. The amplitudes, should be represented by the symbol ‘a’, which will be substituted, by the corresponding value in `amplitudes`.

**Returns**  
\[ \text{sum_expression} : \text{Sympy.Add} \]

Sympy expression of time $t$, where onsets, as a function of $t$, have been symbolically convolved with function $f$, and any function $g$ of corresponding amplitudes.

### Examples

We import some sympy stuff so we can test if we’ve got what we expected

```python
>>> from sympy import DiracDelta, Symbol, Function
>>> from nipy.modalities.fmri.utils import T

>>> evs = events([3,6,9])
>>> evs == DiracDelta(-9 + T) + DiracDelta(-6 + T) + DiracDelta(-3 + T)
True

>>> hrf = Function('hrf')
>>> evs = events([3,6,9], f=hrf)
>>> evs == hrf(-9 + T) + hrf(-6 + T) + hrf(-3 + T)
True

>>> evs = events([3,6,9], amplitudes=[2,1,-1])
>>> evs == -DiracDelta(-9 + T) + 2*DiracDelta(-3 + T) + DiracDelta(-6 + T)
True
```

**nipy.modalities.fmri.utils.fourier_basis(freq)**

**sin and cos Formula for Fourier drift**

The Fourier basis consists of sine and cosine waves of given frequencies.

**Parameters**  
`freq` : sequence of float

Frequencies for the terms in the Fourier basis.

**Returns**  
\[ f : \text{Formula} \]

### Examples

```python
>>> f=fourier_basis([1,2,3])
>>> f.terms
array([cos(2*pi*t), sin(2*pi*t), cos(4*pi*t), sin(4*pi*t), cos(6*pi*t),
      sin(6*pi*t)], dtype=object)

>>> f.mean
_b0*cos(2*pi*t) + _b1*sin(2*pi*t) + _b2*cos(4*pi*t) + _b3*sin(4*pi*t) + _b4*cos(6*pi*t) + _b5*sin(6*pi*t)
```

**nipy.modalities.fmri.utils.interp(times, values, fill=0, name=None, **kw)**

**Generic interpolation function of t given times and values**

Interpolator such that:

\[ f(t) = \text{fill} \]

if \( t < \text{times}[0] \) or \( t > \text{times}[-1] \)
See `scipy.interpolate.interp1d` for details of interpolation types and other keyword arguments. Default is ‘kind’ is linear, making this function, by default, have the same behavior as `linear_interp`.

**Parameters**

- `times`: array-like
  Increasing sequence of times
- `values`: array-like
  Values at the specified times
- `fill`: None or float, optional
  Value on the interval (-np.inf, times[0]). Default 0. If None, raises error outside bounds
- `name`: None or str, optional
  Name of symbolic expression to use. If None, a default is used.
- `**kw`: keyword args, optional
  Passed to `interp1d`

**Returns**

- `f`: sympy expression
  A Function of t.

**Examples**

```python
>>> s = interp([0, 4, 5], [2, 4, 6])
>>> tval = np.array([-0.1, 0.1, 3.9, 4.1, 5.1])
>>> res = lambdify_t(s)(tval)
0 outside bounds by default
>>> np.allclose(res, [0, 2.05, 3.95, 4.2, 0])
True
```

```python
def lambdify_t(expr):
    return sympy.function(expr)
```

```python
def linear_interp(times, values, fill=0, name=None, **kw):
    Interpolator such that:
    f(times[i]) = values[i]
    if t < times[0] or t > times[-1]: f(t) = fill
    This version of the function enforces the ‘linear’ kind of interpolation (argument to `scipy.interpolate.interp1d`).

    **Parameters**
    - `times`: array-like
      Increasing sequence of times
    - `values`: array-like
      Values at the specified times
```

139.2. Functions
fill : None or float, optional
Value on the interval (-np.inf, times[0]). Default 0. If None, raises error outside bounds

name : None or str, optional
Name of symbolic expression to use. If None, a default is used.

**kw : keyword args, optional
passed to interp1d

**Returns**

f : sympy expression
A Function of t.

Examples

```python
>>> s = linear_interp([0, 4, 5.], [2., 4, 6])
>>> tval = np.array([-0.1, 0.1, 3.9, 4.1, 5.1])
>>> res = lambdify_t(s)(tval)
```

0 outside bounds by default

```python
>>> np.allclose(res, [0, 2.05, 3.95, 4.2, 0])
```
True

nipy.modalities.fmri.utils.step_function(times, values, name=None, fill=0)
Right-continuous step function of time t
Function of t such that
f(times[i]) = values[i]
if t < times[0]: f(t) = fill

**Parameters**
times : (N,) sequence
Increasing sequence of times

values : (N,) sequence
Values at the specified times

fill : float
Value on the interval (-np.inf, times[0])

name : str
Name of symbolic expression to use. If None, a default is used.

**Returns**
f_t : sympy expr
Sympy expression f(t) where f is a sympy implemented anonymous function of time that implements the step function. To get the numerical version of the function, use lambdify_t(f_t)

Examples
>>> s = step_function([0,4,5],[2,4,6])
>>> tval = np.array([-0.1,3.9,4.1,5.1])
>>> lam = lambdify_t(s)
>>> lam(tval)
array([ 0.,  2.,  4.,  6.])
140.1 Module: pkg_info

140.2 Functions

nipy.pkg_info.get_pkg_info(pkg_path)
Return dict describing the context of this package

Parameters pkg_path : str
    path containing __init__.py for package

Returns context : dict
    with named parameters of interest

nipy.pkg_info.pkg_commit_hash(pkg_path)
Get short form of commit hash given directory pkg_path

There should be a file called ‘COMMIT_INFO.txt’ in pkg_path. This is a file in INI file format, with at least one section: commit hash, and two variables archive_subst_hash and install_hash. The first has a substitution pattern in it which may have been filled by the execution of git archive if this is an archive generated that way. The second is filled in by the installation, if the installation is from a git archive.

We get the commit hash from (in order of preference):

• A substituted value in archive_subst_hash
• A written commit hash value in ‘install_hash’
• git’s output, if we are in a git repository

If all these fail, we return a not-found placeholder tuple

Parameters pkg_path : str
directory containing package

Returns hash_from : str
    Where we got the hash from - description

hash_str : str
    short form of hash
141.1 Module: testing.decorators

Extend numpy’s decorators to use nipy’s gui and data labels.

141.2 Functions

nipy.testing.decorators.if_datasource(ds, msg)
nipy.testing.decorators.if_example_data(f)
nipy.testing.decorators.if_templates(f)
nipy.testing.decorators.knownfailure(f)
nipy.testing.decorators.make_label_dec(label, ds=None)

Factory function to create a decorator that applies one or more labels.

Parameters

  label : str or sequence
    One or more labels that will be applied by the decorator to the functions it decorates. Labels are attributes of the decorated function with their value set to True.

  ds : str
    An optional docstring for the resulting decorator. If not given, a default docstring is auto-generated.

Returns

  ldec : function
    A decorator.

Examples

>>> slow = make_label_dec('slow')
>>> print slow.__doc__
Labels a test as 'slow'

>>> rare = make_label_dec(['slow','hard'],
...   "Mix labels 'slow' and 'hard' for rare tests")
>>> @rare
... def f(): pass
...
nipy.testing.decorators.needs_review(msg)
    Skip a test that needs further review.

    Parameters  msg : string
        msg regarding the review that needs to be done

nipy.testing.decorators.skip_doctest_if(condition)
    Decorator - mark a function or method for skipping its doctest.

    This decorator allows you to mark a function whose docstring you wish to omit from testing, while preserving the docstring for introspection, help, etc.
142.1 Module: testing.doctester

Inheritance diagram for nipy.testing.doctester:

Custom doctester based on Numpy doctester
To run doctests via nose, you’ll need nosetests nipy/testing/doctester.py --doctest-test, because this file will be identified as containing tests.

142.2 Classes

142.2.1 DocTestSkip

class nipy.testing.doctester.DocTestSkip(obj)
   Bases: object
   Object wrapper for doctests to be skipped.
   __init__(obj)

142.2.2 NipyDocTestFinder

class nipy.testing.doctester.NipyDocTestFinder(Verbose=False,
   parser=<doctest.DocTestParser instance at 0x27383f8>, recurse=True,
   exclude_empty=True)
   Bases: nipy.fixes.numpy.testing.noseclasses.NumpyDocTestFinder
Methods

```
find(obj[, name, module, globs, extraglobs])  Return a list of the DocTests that are defined by the given

__init__(verbose=False, parser=<doctest.DocTestParser instance at 0x27383f8>, recurse=True, exclude_empty=True)
Create a new doctest finder.

The optional argument parser specifies a class or function that should be used to create new DocTest
objects (or objects that implement the same interface as DocTest). The signature for this factory function
should match the signature of the DocTest constructor.

If the optional argument recurse is false, then find will only examine the given object, and not any contained
objects.

If the optional argument exclude_empty is false, then find will include tests for objects with empty doc-
strings.
```

```
find(obj, name=None, module=None, globs=None, extraglobs=None)
Return a list of the DocTests that are defined by the given object’s docstring, or by any of its contained
objects’ docstrings.

The optional parameter module is the module that contains the given object. If the module is not specified
or is None, then the test finder will attempt to automatically determine the correct module. The object’s
module is used:

• As a default namespace, if globs is not specified.

• To prevent the DocTestFinder from extracting DocTests from objects that are imported from other
modules.

• To find the name of the file containing the object.

• To help find the line number of the object within its file.

Contained objects whose module does not match module are ignored.

If module is False, no attempt to find the module will be made. This is obscure, of use mostly in tests: if
module is False, or is None but cannot be found automatically, then all objects are considered to belong to
the (non-existent) module, so all contained objects will (recursively) be searched for doctests.

The globals for each DocTest is formed by combining globs and extraglobs (bindings in extraglobs override
bindings in globs). A new copy of the globals dictionary is created for each DocTest. If globs is not
specified, then it defaults to the module’s __dict__, if specified, or {} otherwise. If extraglobs is not
specified, then it defaults to {}.
```

142.2.3 NipyDoctest

class nipy.testing.doctester.NipyDoctest
    Bases: nipy.fixes.numpy.testing.noseclasses.NumpyDoctest

    Attributes

    enableOpt
    extension
Methods

```
addOptions(parser[, env])  
Add command-line options for this plugin.

add_options(parser[, env])  
Non-camel-case version of func name for backwards compatibility.

afterContext()  
configure(options, config)

doctest_case_class
help()  
Return help for this plugin.

loadTestsFromFile(filename)  
Load doctests from the file.

loadTestsFromModule(module)
makeTest(obj, parent)  
Look for doctests in the given object, which will be a

matches(name)

options(parser[, env])  
Non-camel-case version of func name for backwards compatibility.

out_check_class
prepareTestLoader(loader)  
Capture loader’s suiteClass.

set_test_context(test)

suiteClass
test_finder_class
tolist(val)
wantFile(file)

__init__()  

addOptions (parser, env=None)
Add command-line options for this plugin.
    
The base plugin class adds --with-$name by default, used to enable the plugin.

add_options (parser, env=None)
Non-camel-case version of func name for backwards compatibility.

Warning: DEPRECATED: Do not use this method, use options instead.

afterContext ()
can_configure = False
configure (options, config)
doctest_case_class
    alias of NumpyDocTestCase
doctest_ignore = ['generate_numpy_api.py', 'scons_support.py', 'setupscons.py', 'setup.py']
doctest_optflags = 12
enableOpt = None
enabled = False
extension = None

help ()  
Return help for this plugin. This will be output as the help section of the --with-$name option that enables the plugin.

loadTestsFromFile (filename)
Load doctests from the file.
```

142.2. Classes
Tests are loaded only if filename’s extension matches configured doctest extension.

**loadTestsFromModule** *(module)*

Look for doctests in the given object, which will be a function, method or class.

**matches** *(name)*

```python
name = 'nipydoctest'
```

**options** *(parser, env={...
```python
out_check_class
alias of NipyOutputChecker

**prepareTestLoader** *(loader)*

Capture loader’s suiteClass.

This is used to create test suites from doctest files.
score = 1000
set_test_context (test)
suiteClass
    alias of DoctestSuite
test_finder_class
    alias of NipyDocTestFinder
tolist (val)
wantFile (file)

142.2.4 NipyOutputChecker

class nipy.testing.doctester.NipyOutputChecker
    Bases: nipy.fixes.numpy.testing.noseclasses.NumpyOutputChecker

    Methods

    check_output (want, got, optionflags)
    output_difference (example, got, optionflags)
        Return a string describing the differences between the expected output for a given example (example) and the actual output (got). optionflags is the set of option flags used to compare want and got.

142.3 Functions

nipy.testing.doctester.ignore_dtype (in_str)
    Removes dtype=[dtype] from string in_str

    Parameters in_str : str
        String maybe containing dtype specifier

    Returns out_str : str
        String from which the dtype specifier has been removed.

Examples

>>> arr = np.arange(5, dtype='i2')
Here's the normal repr:
>>> arr
array([0, 1, 2, 3, 4], dtype=int16)

The repr with the dtype bits removed
>>> ignore_dtype(repr(arr))
'array([0, 1, 2, 3, 4])'
>>> ignore_dtype('something(again, dtype=something)')
'something(again)'

Even if there are more closed brackets after the dtype

>>> ignore_dtype('something(again, dtype=something) (1, 2)
'something(again) (1, 2)'

We need the close brackets to match

>>> ignore_dtype('again, dtype=something')
'again, dtype=something'

**nipy.testing.doctester.round_numbers(in_str, precision)**

Replace fp numbers in `in_str` with numbers rounded to `precision`

**Parameters**

- `in_str`: str
  - string possibly containing floating point numbers
- `precision`: int
  - number of decimal places to round to

**Returns**

- `out_str`: str
  - `in_str` with any floating point numbers replaced with same numbers rounded to `precision` decimal places.

**Examples**

>>> round_numbers('A=0.234, B=12.345', 2)
'A=0.23, B=12.35'

Rounds the floating point value as it finds it in the string. This is even true for numbers with exponentials. Remember that:

>>> '%.3f' % 0.3339e-10
'0.000'

This routine will recognize an exponential as something to process, but only works on the decimal part (leaving the exponential part is it is):

>>> round_numbers('(0.3339e-10, "string")', 3)
'(0.334e-10, "string")'

**nipy.testing.doctester.strip_array_repr(in_str)**

Removes array-specific part of repr from string `in_str`

This parser only works on lines that contain only an array repr (and therefore start with `array`, and end with a close parenthesis. To remove dtypes in array reprs that may be somewhere within the line, use the `IGNORE_DTYPE` doctest option.

**Parameters**

- `in_str`: str
  - String maybe containing a repr for an array

**Returns**

- `out_str`: str
  - String from which the array specific parts of the repr have been removed.
Examples

>>> arr = np.arange(5, dtype='i2')

Here’s the normal repr:

>>> arr
array([0, 1, 2, 3, 4], dtype=int16)

The repr with the ‘array’ bits removed:

>>> strip_array_repr(repr(arr))
'\[0, 1, 2, 3, 4\]'
143.1 Module: testing.nosepatch

Monkeypatch nose to accept any callable as a method.

By default, nose’s ismethod() fails for static methods. Once this is fixed in upstream nose we can disable it.

Note: merely importing this module causes the monkeypatch to be applied.

nipy.testing.nosepatch.getTestCaseNames(self, testCaseClass)
    Override to select with selector, unless config.getTestCaseNamesCompat is True
144.1 Module: testing.nosetester

Inheritance diagram for numpy.testing.nosetester:

```
  testing.nosetester.NoseTester --> testing.nosetester.NipyNoseTester
```

Nipy nosetester
Sets doctests to run by default
Use our own doctest plugin (based on that of numpy)

144.2 Class

144.3 NipyNoseTester

```python
class nipy.testing.nosetester.NipyNoseTester (package=None):
    Bases: nipy.fixes.numpy.testing.nosetester.NoseTester

    Numpy-like testing class
    • Removes some numpy-specific excludes
    • Disables numpy’s fierce clearout of module import context for doctests
    • Run doctests by default
```

Methods

```
bench([label, verbose, extra_argv])  Run benchmarks for module using nose.
```
Continued on next page
Table 144.1 – continued from previous page

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>prepare_test_args</td>
<td>Run tests for module using nose.</td>
</tr>
<tr>
<td>test</td>
<td>Run tests for module using nose.</td>
</tr>
</tbody>
</table>

__init__ (package=None)
Test class init

**Parameters**

- **package** : string or module
  If string, gives full path to package If None, extract calling module path Default is None

**bench** (label='fast', verbose=1, extra_argv=None)
Run benchmarks for module using nose.

**Parameters**

- **label** : {'fast', 'full', '', attribute identifier}, optional
  Identifies the benchmarks to run. This can be a string to pass to the nosetests executable with the `-A` option, or one of several special values. Special values are: * 'fast' - the default - which corresponds to the nosetests -A option of 'not slow'.
  - 'full' - fast (as above) and slow benchmarks as in the 'no -A' option to nosetests - this is the same as '':
  - None or '' - run all tests.
- **attribute_identifier** - string passed directly to nosetests as `-A`.
- **verbose** : int, optional
  Verbosity value for benchmark outputs, in the range 1-10. Default is 1.
- **extra_argv** : list, optional
  List with any extra arguments to pass to nosetests.

**Returns**

- **success** : bool
  Returns True if running the benchmarks works, False if an error occurred.

**Notes**

Benchmarks are like tests, but have names starting with “bench” instead of “test”, and can be found under the “benchmarks” sub-directory of the module.

Each NumPy module exposes `bench` in its namespace to run all benchmarks for it.

**Examples**

```python
>>> success = np.lib.bench()
Running benchmarks for numpy.lib ...
... using 562341 items:
unique:
0.11
uniqueld:
0.11
ratio: 1.0
```
nUnique: 56230 == 56230
...
OK

```
>>> success
True
```

```python
excludes = []
```

```python
prepare_test_args(label='fast', verbose=1, extra_argv=None, doctests=False, coverage=False)
```

Run tests for module using nose.

This method does the heavy lifting for the test method. It takes all the same arguments, for details see test.

See Also:

```python
test
```

```python
run_tests_for_module_using_nose.
```

As for numpy tester, except enable tests by default.

**Parameters**

- **label**: {'fast', 'full', '', attribute identifier}, optional
  - Identifies the tests to run. This can be a string to pass to directly the nosetests executable with the '-A' option (an attribute identifier), or one of several special values. Special values are:
  - 'fast' - the default - which corresponds to the nosetests -A option of 'not slow'.
  - 'full' - fast (as above) and slow tests as in the 'no -A' option to nosetests - this is the same as '{}'.
  - None or '' - run all tests.

- **verbose**: int, optional
  - Verbosity value for test outputs, in the range 1-10. Default is 1.

- **extra_argv**: list, optional
  - List with any extra arguments to pass to nosetests.

- **doctests**: bool, optional
  - If True, run doctests in module. Default is True.

- **coverage**: bool, optional
  - If True, report coverage of nipy code. Default is False. (This requires the 'coverage' module:
    ```
<http://nedbatchelder.com/code/modules/coverage.html>`_.
```

**Returns**

- **result**: object
  - Returns the result of running the tests as a nose.result.TextTestResult object.

**Notes**

Each nipy module should expose test in its namespace to run all tests for it. For example, to run all tests for nipy.algorithms:
>>> import nipy.algorithms
>>> nipy.algorithms.test()

144.4 Functions

nipy.testing.nosetester.fpw_opt_str()
Return first-package-wins option string for this version of nose

Versions of nose prior to 1.1.0 needed =True for first-package-wins, versions after won’t accept it.


    Returns fpw_str : str
    Either ‘–first-package-wins’ or ‘–first-package-wins=True’ depending on the nose version we are running.

nipy.testing.nosetester.prepare_imports()
Prepare any imports for testing run

At the moment, we prepare matplotlib by trying to make it use a backend that does not need a display
145.1 Module: utils.arrays

Array utilities

nipy.utils.arrays\texttt{.strides}\_from\texttt{(shape, dtype, order='\textsc{C}')}

Return strides as for continuous array \texttt{shape} and given \texttt{dtype}

\begin{itemize}
  \item \textbf{Parameters} \hspace{1cm} \texttt{shape}: sequence
  \hspace{1cm} shape of array to calculate strides from
  \hspace{1cm} \texttt{dtype}: dtype-like
  \hspace{1cm} dtype specifier for array
  \hspace{1cm} \texttt{order}: \{\textsc{C}, \textsc{F}\}, optional
  \hspace{1cm} whether array is C or FORTRAN ordered

  \item \textbf{Returns} \hspace{1cm} \texttt{strides}: tuple
  \hspace{1cm} sequence length \texttt{len(shape)} giving strides for continuous array with given \texttt{shape}, \texttt{dtype} and \texttt{order}
\end{itemize}

\textbf{Examples}

\begin{verbatim}
>>> strides_from((2, 3, 4), 'i4')
(48, 16, 4)
>>> strides_from((3, 2), np.float)
(16, 8)
>>> strides_from((5, 4, 3), np.bool, order='F')
(1, 5, 20)
\end{verbatim}
146.1 Module: `utils.compat3`

Routines for Python 3 compatibility

These are in addition to the nibabel.py3k routines.

```python
nipy.utils.compat3.open4csv(fname, mode)
```

Open filename `fname` for CSV IO in read or write `mode`

**Parameters**

- `fname`: str
  - filename to open
- `mode`: {'r', 'w'}
  - Mode to open file. Don’t specify binary or text modes; we need to chose these according to python version.

**Returns**

- `fobj`: file object
  - open file object; needs to be closed by the caller
147.1 Module: utils.perlpie

Perform a global search and replace on the current directory recursively.

This a small python wrapper around the `perl -p -i -e` functionality. I strongly recommend running `perlpie` on files under source control. In this way it’s easy to track your changes and if you discover your regular expression was wrong you can easily revert. I also recommend using `grin` to test your regular expressions before running `perlpie`.

147.1.1 Parameters

**regex** [regular expression] Regular expression matching the string you want to replace

**newstring** [string] The string you would like to replace the oldstring with. Note this is not a regular expression but the exact string. One exception to this rule is the at symbol @. This has special meaning in perl, so you need an escape character for this. See Examples below.

147.1.2 Requires

perl : The underlying language we’re using to perform the search and replace.

grin : Grin is a tool written by Robert Kern to wrap `grep` and `find` with python and easier command line options.

147.1.3 Examples

Replace all occurrences of foo with bar:

`perlpie foo bar`

Replace numpy.testing with nipy’s testing framework:

`perlpie 'from\s+numpy\testing.*' 'from nipy.testing import *'`

Replace all @slow decorators in my code with @dec.super_slow. Here we have to escape the @ symbol which has special meaning in perl:

`perlpie '\@slow' '\@dec.super_slow'`

Remove all occurrences of importing make_doctest_suite:
perlpie 'from\snipy\utils\testutils\*make_doctest_suite'

147.2 Functions

nipy.utils.perlpie.check_deps()
nipy.utils.perlpie.main()
nipy.utils.perlpie.perl_dash_pie(\texttt{oldstr}, \texttt{newstr}, \texttt{dry\_run=None})

Use perl to replace the oldstr with the newstr.

Example

\# To replace all occurrences of \texttt{import numpy as N} with \texttt{import numpy as np} from nipy.utils import perlpie perlpie.perl_dash_pie('imports+numpys+ass+N', 'import numpy as np') grind | xargs perl -pi -e 's/imports+numpys+ass+N/import numpy as np/g'

nipy.utils.perlpie.print_extended_help(\texttt{option}, \texttt{opt\_str}, \texttt{value}, \texttt{parser}, *\texttt{args}, **\texttt{kwargs})
148.1 Module: `utils.skip_test`

Utilities to skip tests

```python
nipy.utils.skip_test.skip_if_running_nose(msg=None)
```

Raise a SkipTest if we appear to be running the nose test loader.

**Parameters**

- `msg` : string, optional

  The message issued when SkipTest is raised
Part VI

Publications
Part VII

NIPY License Information
SOFTWARE LICENSE

Except where otherwise noted, all NIPY software is licensed under a revised BSD license.

See our Licensing page for more details.
Except where otherwise noted, all NIPY documentation is licensed under a Creative Commons Attribution 3.0 License. All code fragments in the documentation are licensed under our software license.


[R10] 7. (a) Golub and C. F. Van Loan, _Matrix Computations_.


nipy.algorithms.clustering.bgmm, 149
nipy.algorithms.clustering.ggmixture, 167
nipy.algorithms.clustering.gmm, 173
nipy.algorithms.clustering.hierarchical_clustering, 181
nipy.algorithms.clustering.imm, 195
nipy.algorithms.clustering.utils, 213
nipy.algorithms.clustering.von_mises_fisher_mixture, 215
nipy.algorithms.diagnostics.screens, 221
nipy.algorithms.diagnostics.timediff, 223
nipy.algorithms.diagnostics.tsdiffplot, 225
nipy.algorithms.fwhm, 227
nipy.algorithms.graph.bipartite_graph, 231
nipy.algorithms.graph.field, 235
nipy.algorithms.graph.forest, 245
nipy.algorithms.graph.graph, 255
nipy.algorithms.interpolation, 265
nipy.algorithms.kernel_smooth, 267
nipy.algorithms.optimize, 271
nipy.algorithms.registration.affine, 273
nipy.algorithms.registration.chain_transform, 281
nipy.algorithms.registration.groupwise_registration, 283
nipy.algorithms.registration.histogram_registration, 291
nipy.algorithms.registration.optimizer, 295
nipy.algorithms.registration.polyaffine, 297
nipy.algorithms.registration.resample, 299
nipy.algorithms.registration.similarity, 301
nipy.algorithms.registration.transform, 307
nipy.algorithms.resample, 309
nipy.algorithms.segmentation.brain_segmentation, 311
nipy.algorithms.segmentation.segmentation, 313
nipy.algorithms.statistics.bench.bench_intvol, 317
nipy.algorithms.statistics.empirical_pvalue, 319
nipy.algorithms.statistics.formula.formulae, 325
nipy.algorithms.statistics.mixed_effects_stat, 423
nipy.algorithms.statistics.models.family.family, 429
nipy.algorithms.statistics.models.family.links, 439
nipy.algorithms.statistics.models.family.varfuncs, 445
nipy.algorithms.statistics.models.glm, 447
nipy.algorithms.statistics.models.model, 451
nipy.algorithms.statistics.models.nlsmodel, 457
nipy.algorithms.statistics.models.regression, 461
nipy.algorithms.statistics.models.setupscons, 481
nipy.algorithms.statistics.models.utils, 483
nipy.algorithms.statistics.onesample, 485
nipy.algorithms.statistics.rft, 487
nipy.algorithms.statistics.utils, 499
nipy.algorithms.utils.fast_distance, 501
nipy.algorithms.utils.matrices, 503
nipy.algorithms.utils.pca, 507
nipy.core.image.image, 511
nipy.core.image.image_list, 523
nipy.core.image.image_spaces, 527
nipy.core.reference.array_coords, 533
nipy.core.reference.coordinate_map, 537
nipy.core.reference.coordinate_system, 561
nipy.core.reference.slices, 567
nipy.core.reference.spaces, 571
nipy.core.utils.generators, 579
nipy.interfaces.matlab, 583
nipy.interfaces.spm, 585
nipy.io.files, 587
nipy.io.nii_ref, 591
nipy.labs.bindings.benchmarks.bench_numpy, 597
nipy.labs.datasets.converters, 599
nipy.labs.datasets.transforms.affine_transform, 601
nipy.labs.datasets.transforms.affine_utils, 603
nipy.labs.datasets.transforms.transform, 605
nipy.labs.datasets.volumes.volume_data, 609
nipy.labs.datasets.volumes.volume_field, 613
nipy.labs.datasets.volumes.volume_grid, 617
nipy.labs.datasets.volumes.volume_img, 621
nipy.labs.glm.benchmarks.bench_glm, 625
nipy.labs.glm.glm, 627
nipy.labs.group.displacement_field, 629
nipy.labs.group.mixed_effects, 633
nipy.labs.group.permutation_test, 635
nipy.labs.group.spatial_relaxation_onesample, 645
nipy.labs.labs.spatial_models.bayesianstructural_analysis, 655
nipy.labs.labs.spatial_models.bsa_io, 661
nipy.labs.labs.spatial_models.discrete_domain, 663
nipy.labs.labs.spatial_models.hierarchical_parcellation, 673
nipy.labs.labs.spatial_models.hroi, 675
nipy.labs.labs.spatial_models.mroi, 685
nipy.labs.labs.spatial_models.parcel_io, 693
nipy.labs.labs.spatial_models.parcellation, 697
nipy.labs.labs.spatial_models.structural_bfls, 701
nipy.labs.statistical_mapping, 705
nipy.labs.utils.random_threshold, 709
nipy.labs.utils.reproducibility_measures, 715
nipy.labs.utils.simul_multisubject_fmri_dataset, 723
nipy.labs.utils.zscore, 727
nipy.viz_tools.activation_maps, 729
nipy.viz_tools.anat_cache, 733
nipy.viz_tools.cm, 735
nipy.viz_tools.coord_tools, 737
nipy.viz_tools.maps_3d, 739
nipy.viz_tools.slicers, 743
nipy.viz_tools.test.test_activation_maps, 761
nipy.viz_tools.test.test_cm, 763
nipy.viz_tools.test.test_coord_tools, 765
nipy.viz_tools.test.test_edge_detect, 767
nipy.viz_tools.test.test_slicers, 769
nipy.modalities.fmri.design, 771
nipy.modalities.fmri.design_matrix, 775
nipy.modalities.fmri.experimental_paradigm, 779
nipy.modalities.fmri.fmrirstat.delay, 787
nipy.modalities.fmri.fmrirstat.hrf, 789
nipy.modalities.fmri.fmrirstat.invert, 791
nipy.modalities.fmri.fmrirstat.model, 793
nipy.modalities.fmri.glm, 803
nipy.modalities.fmri.hemodynamic_models, 809
nipy.modalities.fmri.hrf, 813
nipy.modalities.fmri.spm.correlation, 815
nipy.modalities.fmri.spm.model, 817
nipy.modalities.fmri.spm.reml, 819
nipy.modalities.fmri.spm.trace, 821
nipy.modalities.fmri.utils, 823
nipy.pkg_info, 831
nipy.testing.decorators, 833
nipy.testing.doctester, 835
nipy.testing.nosepatch, 843
nipy.testing.nosetester, 845
nipy.utils.arrays, 849
nipy.utils.compat3, 851
nipy.utils.perlpie, 853
nipy.utils.skip_test, 855
nipy.algorithms.clustering.bgmm, 149
nipy.algorithms.clustering.ggmixture, 167
nipy.algorithms.clustering.gmm, 173
nipy.algorithms.clustering.hierarchical_clustering, 181
nipy.algorithms.clustering.imm, 195
nipy.algorithms.clustering.von_mises_fisher_mixture, 215
nipy.algorithms.diagnostics.screens, 221
nipy.algorithms.diagnostics.timediff, 223
nipy.algorithms.diagnostics.tsdiffplot, 225
nipy.algorithms.fwhm, 227
nipy.algorithms.graph.bipartite_graph, 231
nipy.algorithms.graph.field, 235
nipy.algorithms.graph.forest, 245
nipy.algorithms.graph.graph, 255
nipy.algorithms.interpolation, 265
nipy.algorithms.kernel_smooth, 267
nipy.algorithms.optimize, 271
nipy.algorithms.registration.affine, 273
nipy.algorithms.registration.chain_transform, 281
nipy.algorithms.registration.groupwise_registration, 283
nipy.algorithms.registration.histogram_registration, 291
nipy.algorithms.registration.optimizer, 295
nipy.algorithms.registration.polyaffine, 297
nipy.algorithms.registration.resample, 299
nipy.algorithms.registration.similarity, 301
nipy.algorithms.registration.transform, 307
nipy.algorithms.resample, 309
nipy.algorithms.segmentation.bench.bench_intvol, 311
nipy.algorithms.segmentation.segmentation, 313
nipy.algorithms.statistics.bench.bench_intvol, 317
nipy.algorithms.statistics.empirical_pvalue, 319
nipy.algorithms.statistics.models.family.family, 325
nipy.algorithms.statistics.mixed_effects_stat, 423
nipy.algorithms.statistics.models.family.family, 429
nipy.algorithms.statistics.models.family.links, 439
nipy.algorithms.statistics.models.family.varfuncs, 445
nipy.algorithms.statistics.models.glm, 447
nipy.algorithms.statistics.models.model, 451
nipy.algorithms.statistics.models.nlsmodel, 457
nipy.algorithms.statistics.models.regression, 461
nipy.algorithms.statistics.models.setupscons, 481
nipy.algorithms.statistics.models.statsmodels, 481
nipy.algorithms.statistics.models.utils, 483
nipy.algorithms.statistics.onesample, 485
nipy.algorithms.statistics.rft, 487
nipy.algorithms.statistics.utils, 499
nipy.algorithms.utils.fast_distance, 501
nipy.algorithms.utils.matrices, 503
nipy.algorithms.utils.pca, 507
nipy.core.image.image, 511
nipy.core.image.image_list, 523
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>nipy.core.image.image_spaces</td>
<td>527</td>
</tr>
<tr>
<td>nipy.core.reference.array_coords</td>
<td>533</td>
</tr>
<tr>
<td>nipy.core.reference.coordinate_map</td>
<td>537</td>
</tr>
<tr>
<td>nipy.core.reference.coordinate_system</td>
<td>561</td>
</tr>
<tr>
<td>nipy.core.reference.coordinates</td>
<td>571</td>
</tr>
<tr>
<td>nipy.core.reference.constraint_map</td>
<td>575</td>
</tr>
<tr>
<td>nipy.core.reference.coordinate_system</td>
<td>561</td>
</tr>
<tr>
<td>nipy.core.reference.slices</td>
<td>567</td>
</tr>
<tr>
<td>nipy.core.reference.spaces</td>
<td>571</td>
</tr>
<tr>
<td>nipy.core.utils.generators</td>
<td>579</td>
</tr>
<tr>
<td>nipy.IO.files</td>
<td>587</td>
</tr>
<tr>
<td>nipy.IO.nifti_ref</td>
<td>591</td>
</tr>
<tr>
<td>nipy.labs.bindings.benchmarks.bench_num</td>
<td>595</td>
</tr>
<tr>
<td>nipy.labs.datasets.converters</td>
<td>599</td>
</tr>
<tr>
<td>nipy.labs.datasets.transforms.affine_transform</td>
<td>601</td>
</tr>
<tr>
<td>nipy.labs.datasets.transforms.affine_utils</td>
<td>603</td>
</tr>
<tr>
<td>nipy.labs.datasets.transforms.transform</td>
<td>605</td>
</tr>
<tr>
<td>nipy.labs.datasets.volumes.volume_data</td>
<td>609</td>
</tr>
<tr>
<td>nipy.labs.datasets.volumes.volume_field</td>
<td>613</td>
</tr>
<tr>
<td>nipy.labs.datasets.volumes.volume_grid</td>
<td>617</td>
</tr>
<tr>
<td>nipy.labs.datasets.volumes.volume_img</td>
<td>621</td>
</tr>
<tr>
<td>nipy.labs.glm.benchmarks.bench_glm</td>
<td>625</td>
</tr>
<tr>
<td>nipy.labs.glm.glm</td>
<td>627</td>
</tr>
<tr>
<td>nipy.labs.group.displacement_field</td>
<td>629</td>
</tr>
<tr>
<td>nipy.labs.group.mixed_effects</td>
<td>633</td>
</tr>
<tr>
<td>nipy.labs.group.permutation_test</td>
<td>635</td>
</tr>
<tr>
<td>nipy.labs.group.spatial_restoration_onesample</td>
<td>637</td>
</tr>
<tr>
<td>nipy.labs.mask</td>
<td>651</td>
</tr>
<tr>
<td>nipy.labs.spatial_models.bayesian_analysis</td>
<td>653</td>
</tr>
<tr>
<td>nipy.labs.spatial_models.bsa_io</td>
<td>661</td>
</tr>
<tr>
<td>nipy.labs.spatial_models.discrete_domain</td>
<td>663</td>
</tr>
<tr>
<td>nipy.labs.spatial_models.hierarchical_parcellation</td>
<td>673</td>
</tr>
<tr>
<td>nipy.labs.spatial_models.hroi</td>
<td>675</td>
</tr>
<tr>
<td>nipy.labs.spatial_models.mroi</td>
<td>685</td>
</tr>
<tr>
<td>nipy.labs.spatial_models.parcellation</td>
<td>693</td>
</tr>
<tr>
<td>nipy.labs.spatial_models.structural_bfls</td>
<td>697</td>
</tr>
<tr>
<td>nipy.labs.spatial_models.statistical_mapping</td>
<td>701</td>
</tr>
<tr>
<td>nipy.labs.stats.random_threshold</td>
<td>709</td>
</tr>
<tr>
<td>nipy.labs.utils.reproducibility_measures</td>
<td>709</td>
</tr>
<tr>
<td>nipy.labs.utils.random_threshold</td>
<td>709</td>
</tr>
<tr>
<td>nipy.labs.utils.simul_multisubject_fmri_dataset</td>
<td>723</td>
</tr>
<tr>
<td>nipy.labs.utils.zscore</td>
<td>727</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.activation_maps</td>
<td>729</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.anat_cache</td>
<td>733</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.cm</td>
<td>735</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.coord_tools</td>
<td>737</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.maps_3d</td>
<td>739</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.slicers</td>
<td>743</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.test.test_activation_maps</td>
<td>761</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.test.test_cm</td>
<td>763</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.test.test_coord_tools</td>
<td>765</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.test.test_edge_detect</td>
<td>767</td>
</tr>
<tr>
<td>nipy.labs.viz_tools.test.test_slicers</td>
<td>769</td>
</tr>
<tr>
<td>nipy.modalities.fmri.design</td>
<td>771</td>
</tr>
<tr>
<td>nipy.modalities.fmri.design_matrix</td>
<td>775</td>
</tr>
<tr>
<td>nipy.modalities.fmri.experimental_paradigm</td>
<td>779</td>
</tr>
<tr>
<td>nipy.modalities.fmri.fmri</td>
<td>783</td>
</tr>
<tr>
<td>nipy.modalities.fmri.fmristat.delay</td>
<td>787</td>
</tr>
<tr>
<td>nipy.modalities.fmri.fmristat.hrf</td>
<td>789</td>
</tr>
<tr>
<td>nipy.modalities.fmri.fmristat.invert</td>
<td>791</td>
</tr>
<tr>
<td>nipy.modalities.fmri.fmristat.model</td>
<td>793</td>
</tr>
<tr>
<td>nipy.modalities.fmri.fmristat.outputters</td>
<td>799</td>
</tr>
<tr>
<td>nipy.modalities.fmri.glm</td>
<td>803</td>
</tr>
<tr>
<td>nipy.modalities.fmri.hemodynamic_models</td>
<td>809</td>
</tr>
<tr>
<td>nipy.modalities.fmri.hrf</td>
<td>813</td>
</tr>
<tr>
<td>nipy.modalities.fmri.spm.correlation</td>
<td>815</td>
</tr>
<tr>
<td>nipy.modalities.fmri.spm.model</td>
<td>817</td>
</tr>
<tr>
<td>nipy.modalities.fmri.spm.reml</td>
<td>819</td>
</tr>
<tr>
<td>nipy.modalities.fmri.spm.trace</td>
<td>821</td>
</tr>
<tr>
<td>nipy.modalities.fmri.utils</td>
<td>823</td>
</tr>
<tr>
<td>nipy.pkg_info</td>
<td>831</td>
</tr>
<tr>
<td>nipy.testing.decorators</td>
<td>833</td>
</tr>
<tr>
<td>nipy.testing.doctester</td>
<td>835</td>
</tr>
<tr>
<td>nipy.testing.nosepatch</td>
<td>843</td>
</tr>
<tr>
<td>nipy.testing.nosetester</td>
<td>845</td>
</tr>
<tr>
<td>nipy.utils.arrays</td>
<td>849</td>
</tr>
<tr>
<td>nipy.utils.compat</td>
<td>851</td>
</tr>
<tr>
<td>nipy.utils.perlpie</td>
<td>853</td>
</tr>
<tr>
<td>nipy.utils.skip_test</td>
<td>855</td>
</tr>
</tbody>
</table>

874 Python Module Index
Symbols

__init__() (nipy.algorithms.clustering.bgmm.BGMM method), 151
__init__() (nipy.algorithms.clustering.bgmm.VBGMM method), 157
__init__() (nipy.algorithms.clustering.ggmixture.GGM method), 168
__init__() (nipy.algorithms.clustering.ggmixture.GGGM method), 171
__init__() (nipy.algorithms.clustering.ggmixture.Gamma method), 172
__init__() (nipy.algorithms.clustering.gmm.GMM method), 174
__init__() (nipy.algorithms.clustering.gmm.GridDescriptor method), 178
__init__() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 183
__init__() (nipy.algorithms.clustering.imm.IMM method), 196
__init__() (nipy.algorithms.clustering.imm.MixedIMM method), 204
__init__() (nipy.algorithms.clustering.von_mises_fisher_mixture.VonMisesFisher method), 216
__init__() (nipy.algorithms.fwhm.ReselImage method), 228
__init__() (nipy.algorithms.fwhm.Resels method), 229
__init__() (nipy.algorithms.graph.bipartite_graph.BipartiteGraph method), 232
__init__() (nipy.algorithms.graph.field.Field method), 236
__init__() (nipy.algorithms.graph.forest.Forest method), 247
__init__() (nipy.algorithms.graph.Graph method), 256
__init__() (nipy.algorithms.graph.WeightedGraph method), 258
__init__() (nipy.algorithms.interpolation.ImageInterpolator method), 265
__init__() (nipy.algorithms.kernel_smooth.LinearFilter method), 267
__init__() (nipy.algorithms.registration.affine.Affine method), 273
__init__() (nipy.algorithms.registration.affine.Affine2D method), 274
__init__() (nipy.algorithms.registration.affine.Rigid method), 275
__init__() (nipy.algorithms.registration.affine.Rigid2D method), 276
__init__() (nipy.algorithms.registration.affine.Similarity method), 277
__init__() (nipy.algorithms.registration.affine.Similarity2D method), 278
__init__() (nipy.algorithms.registration.chain_transform.ChainTransform method), 281
__init__() (nipy.algorithms.registration.groupwise_registration.FmriRealign method), 284
__init__() (nipy.algorithms.registration.groupwise_registration.Image4d method), 285
__init__() (nipy.algorithms.registration.groupwise_registration.Realign4d method), 286
__init__() (nipy.algorithms.registration.groupwise_registration.Realign4dAlgorithm method), 288
__init__() (nipy.algorithms.registration.histogram_registration.HistogramRegistration method), 292
__init__() (nipy.algorithms.registration.polyaffine.PolyAffine method), 297
__init__() (nipy.algorithms.registration.similarity_measures.CorrelationCoefficient method), 302
__init__() (nipy.algorithms.registration.similarity_measures.CorrelationRatio method), 302
__init__() (nipy.algorithms.registration.similarity_measures.DiscreteParzen method), 303
__init__() (nipy.algorithms.registration.similarity_measures.MutualInformation method), 304
__init__() (nipy.algorithms.registration.similarity_measures.NormalizedMutualInformation method), 304
__init__() (nipy.algorithms.registration.similarity_measures.ParzenMutualInformation method), 305
__init__() (nipy.algorithms.registration.similarity_measures.SupervisedLikelihood method), 307
__init__() (nipy.algorithms.registration.transform.Transform method), 305
__init__() (nipy.algorithms.registration.brain_segmentation.brain_segmentation method), 308
__init__() (nipy.algorithms.registration.brain_segmentation.brain_segmentation.BrainT1Segmentation method), 311
__init__() (nipy.algorithms.registration.brain_segmentation.brain_segmentation.Segmentation method), 314
__init__() (nipy.algorithms.statistics.empirical_pvalue.NormalEmpiricalNull method), 43, 320
__init__() (nipy.algorithms.statistics.formula.formulae.Beta method), 329
__init__() (nipy.algorithms.statistics.formula.formulae.Factor method), 355
__init__() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 359
__init__() (nipy.algorithms.statistics.formula.formulae.Formula method), 385
__init__() (nipy.algorithms.statistics.formula.formulae.RandomEffects method), 388
__init__() (nipy.algorithms.statistics.formula.formulae.Term method), 392
__init__() (nipy.algorithms.statistics.models.family.family.Binomial method), 424
__init__() (nipy.algorithms.statistics.models.family.family.Family method), 430
__init__() (nipy.algorithms.statistics.models.family.family.Gamma method), 432
__init__() (nipy.algorithms.statistics.models.family.family.Gaussian method), 434
__init__() (nipy.algorithms.statistics.models.family.family.InverseGaussian method), 436
__init__() (nipy.algorithms.statistics.models.family.family.Poisson method), 438
__init__() (nipy.algorithms.statistics.models.family.varfuncs.Binomial method), 446
__init__() (nipy.algorithms.statistics.models.family.varfuncs.Power method), 446
__init__() (nipy.algorithms.statistics.models.family.varfuncs.VarianceFunction method), 446
__init__() (nipy.algorithms.statistics.models.glm.Model method), 448
__init__() (nipy.algorithms.statistics.models.model.FContrastResults method), 451
__init__() (nipy.algorithms.statistics.models.model.LikelihoodModel method), 453
__init__() (nipy.algorithms.statistics.models.model.LikelihoodModelResults method), 456
__init__() (nipy.algorithms.statistics.models.model.Model method), 456
__init__() (nipy.algorithms.statistics.models.model.TContrastResults method), 456
__init__() (nipy.algorithms.statistics.models.nlsmodel.NLSModel method), 457
__init__() (nipy.algorithms.statistics.models.regression.AREstimator method), 463
__init__() (nipy.algorithms.statistics.models.regression.ARMModel method), 464
__init__() (nipy.algorithms.statistics.models.regression.GLSModel method), 467
__init__() (nipy.algorithms.statistics.models.regression.OLSModel method), 469
__init__() (nipy.algorithms.statistics.models.regression.RegressionResults method), 472
__init__() (nipy.algorithms.statistics.models.regression.WLSModel method), 476
__init__() (nipy.algorithms.statistics.rft.ChiBarSquared method), 488
__init__() (nipy.algorithms.statistics.rft.ChiSquared method), 488
__init__() (nipy.algorithms.statistics.rft.ECcone method), 489
__init__() (nipy.algorithms.statistics.rft.ECquasi method), 490
__init__() (nipy.algorithms.statistics.rft.FStat method), 492
__init__() (nipy.algorithms.statistics.rft.fnsum method), 495
__init__() (nipy.algorithms.statistics.rft.IntrinsicVolumes method), 492
__init__() (nipy.algorithms.statistics.rft.MultilinearForm method), 493
__init__() (nipy.algorithms.statistics.rft.OneSidedF method), 494
__init__() (nipy.algorithms.statistics.rft.Roy method), 494
__init__() (nipy.algorithms.statistics.rft.TStat method), 495
__init__() (nipy.algorithms.statistics.rft.fnsum method), 495
__init__() (nipy.core.image.image.Image method), 512
__init__() (nipy.core.image.image.SliceMaker method), 516
Neuroimaging in Python Documentation, Release 0.3.0

Index
anti_symmeterize() (nipy.algorithms.graph.field.Field method), 237
anti_symmeterize() (nipy.algorithms.graph.forest.Forest method), 247
anti_symmeterize() (nipy.algorithms.graph.graph.WeightedGraph method), 258
apart() (nipy.algorithms.statistics.formula.formulae.Beta method), 329
apart() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 359
apart() (nipy.algorithms.statistics.formula.formulae.Term method), 392
append_io_dim() (in module nipy.core.reference.coordinate_map), 551
apply() (nipy.algorithms.registration.affine.Affine method), 273
apply() (nipy.algorithms.registration.affine.Affine2D method), 274
apply() (nipy.algorithms.registration.affine.Rigid method), 275
apply() (nipy.algorithms.registration.affine.Rigid2D method), 276
apply() (nipy.algorithms.registration.affine.Similarity method), 277
apply() (nipy.algorithms.registration.affine.Similarity2D method), 278
apply() (nipy.algorithms.registration.chain_transform.ChainTransform method), 282
apply() (nipy.algorithms.registration.polyaffine.PolyAffine method), 298
apply() (nipy.algorithms.registration.transform.Transform method), 308
apply_affine() (in module nipy.labs.datasets.transforms.affine_utils), 603
AR1 (class in nipy.modalities.fmri.fmristat.model), 793
ar_bias_correct() (in module nipy.algorithms.statistics.models.regression), 478
ar_bias_corrector() (in module nipy.algorithms.statistics.models.regression), 478
ARcomponents() (in module nipy.modalities.fmri.spm.correlation), 815
ARcovariance() (in module nipy.modalities.fmri.spm.correlation), 815
area() (nipy.labs.spatial_models.discrete_domain.MeshDomain method), 665
AREstimator (class in nipy.algorithms.statistics.models.regression), 461
args (nipy.algorithms.statistics.formula.formulae.Beta attribute), 359
args (nipy.algorithms.statistics.formula.formulae.Term attribute), 392
args (nipy.core.reference.coordinate_map.AxisError attribute), 544
args (nipy.core.reference.coordinate_map.CoordMapMakerError attribute), 547
args (nipy.core.reference.coordinate_system.CoordinateSystemError attribute), 564
args (nipy.core.reference.coordinate_system.CoordSysMakerError attribute), 562
args (nipy.core.reference.stages.AffineError attribute), 571
args (nipy.core.reference.stages.AxesError attribute), 572
args (nipy.core.reference.stages.SpaceError attribute), 572
args (nipy.core.reference.stages.SpaceTypeError attribute), 572
args (nipy.io.nifti_ref.NiftiError attribute), 593
args (nipy.labs.datasets.transforms.transform.CompositionError attribute), 605
args_cnc() (nipy.algorithms.statistics.formula.formulae.Beta method), 329
args_cnc() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 360
args_cnc() (nipy.algorithms.statistics.formula.formulae.Term method), 393
ARModel (class in nipy.algorithms.statistics.models.regression), 463
array_affine_coord() (in module nipy.labs.spatial_models.discrete_domain), 669
ArrayCoordMap (class in nipy.core.reference.array_coords), 533
as_affine() (nipy.algorithms.registration.affine.Affine method), 273
as_affine() (nipy.algorithms.registration.affine.Affine2D method), 274
as_affine() (nipy.algorithms.registration.affine.Rigid method), 275
as_affine() (nipy.algorithms.registration.affine.Rigid2D method), 276
as_affine() (nipy.algorithms.registration.affine.Similarity method), 277
as_affine() (nipy.algorithms.registration.affine.Similarity2D method), 278
as_base_exp() (nipy.algorithms.statistics.formula.formulae.Beta method), 330
as_base_exp() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 360
as_base_exp() (nipy.algorithms.statistics.formula.formulae.Term method), 393
as_coeff_Add() (nipy.algorithms.statistics.formula.formulae.Beta method), 330
Index 879
as_powers_dict() (nipy.algorithms.statistics.formula.formulae.Beta method), 336
as_powers_dict() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 366
as_powers_dict() (nipy.algorithms.statistics.formula.formulae.Term method), 399
as_real_imag() (nipy.algorithms.statistics.formula.formulae.Beta method), 336
as_real_imag() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 366
as_real_imag() (nipy.algorithms.statistics.formula.formulae.Term method), 399
as_terms() (nipy.algorithms.statistics.formula.formulae.Beta method), 336
as_terms() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 367
as_terms() (nipy.algorithms.statistics.formula.formulae.Term method), 399
as_tuple() (nipy.core.reference.spaces.XYZSpace method), 573
as_volume_img() (in module nipy.labs.datasets.converters), 599
as_volume_img() (nipy.labs.datasets.volumes.volume_data method), 610
as_volume_img() (nipy.labs.datasets.volumes.volume_field method), 59, 614
as_volume_img() (nipy.labs.datasets.volumes.volume_grid method), 618
as_volume_img() (nipy.labs.datasets.volumes.volume_img method), 622
as_xz_image() (in module nipy.core.image.image_spaces), 528
assumptions0 (nipy.algorithms.statistics.formula.formulae.Beta attribute), 336
assumptions0 (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 367
assumptions0 (nipy.algorithms.statistics.formula.formulae.Term attribute), 399
atoms() (nipy.algorithms.statistics.formula.formulae.Beta method), 336
atoms() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 367
atoms() (nipy.algorithms.statistics.formula.formulae.Term method), 399
autocrop_img() (in module nipy.labs.viz_tools.maps_3d), 739
average_link_graph() (in module nipy.algorithms.clustering.hierarchical_clustering), 189
average_link_graph_segment() (in module nipy.algorithms.clustering.hierarchical_clustering), 189
average_log_like() (nipy.algorithms.clustering.bgmm.BGMM method), 151
average_log_like() (nipy.algorithms.clustering.bgmm.VBGMM method), 157
average_log_like() (nipy.algorithms.clustering.gmm.GMM method), 174
average_log_like() (nipy.algorithms.clustering.imm.IMM method), 196
average_log_like() (nipy.algorithms.clustering.imm.MixedIMM method), 204
averaged_loss() (nipy.algorithms.registration.similarity_measures.CorrelationCoefficient method), 302
averaged_loss() (nipy.algorithms.registration.similarity_measures.CorrelationRatio method), 302
averaged_loss() (nipy.algorithms.registration.similarity_measures.CorrelationRatioL1 method), 303
averaged_loss() (nipy.algorithms.registration.similarity_measures.DiscreteParzenMutualInformation method), 303
averaged_loss() (nipy.algorithms.registration.similarity_measures.MutualInformation method), 304
averaged_loss() (nipy.algorithms.registration.similarity_measures.NormalizedMutualInformation method), 304
averaged_loss() (nipy.algorithms.registration.similarity_measures.ParzenMutualInformation method), 305
averaged_loss() (nipy.algorithms.registration.similarity_measures.SupervisedLikelihoodRatio method), 305
ax3d() (nipy.core.reference.coordinate_map attribute), 551
axis0_generator() (in module nipy.modalities.fmri.fmri), 785
B
ball_search() (in module nipy.algorithms.statistics.rft), 513
BaseSlicer (class in nipy.labs.viz_tools.slicers), 743
BaseStackedSlicer (class in nipy.labs.viz_tools.slicers), 746
bayes_factor() (nipy.algorithms.clustering.bgmm.BGMM method), 151
bayes_factor() (nipy.algorithms.clustering.bgmm.VBGMM method), 157
bayes_factor() (nipy.algorithms.clustering.imm.IMM method), 196
bayes_factor() (nipy.algorithms.clustering.imm.MixedIMM method), 204
bench() (nipy.testing.nosetester.NipyNoseTester method), 846
bench_copy_vector_contiguous() (in module nipy.labs.bindings.benchmarks.bench_numpy), 881
Index 883
cut_redundancies() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 184

cut_redundancies() (nipy.algorithms.graph.field.Field method), 238

cut_redundancies() (nipy.algorithms.graph.forest.Forest method), 248

cut_redundancies() (nipy.algorithms.graph.graph.WeightedGraph method), 259

CutAxes (class in nipy.labs.viz_tools.slicers), 748

data_generator() (in module nipy.core.utils.generators), 579

data_scaling() (in module nipy.modalities.fmri.glm), 808

decompose2d() (in module nipy.algorithms.statistics.utils), 499

decompose3d() (in module nipy.algorithms.statistics.utils), 499

def_model (nipy.labs.statistical_mapping.LinearModel attribute), 705

def_niter (nipy.labs.statistical_mapping.LinearModel attribute), 705

default_assumptions (nipy.algorithms.statistics.formula.formulae.Formula attribute), 340

default_assumptions (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 371

default_assumptions (nipy.algorithms.statistics.formula.formulae.Term attribute), 403

define() (in module nipy.algorithms.statistics.formula.formulae), 418

define() (in module nipy.modalities.fmri.utils), 825

define_graph_attributes() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 184

define_graph_attributes() (nipy.algorithms.graph.forest.Forest method), 248

degrees() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 184

degrees() (nipy.algorithms.graph.field.Field method), 238

degrees() (nipy.algorithms.graph.forest.Forest method), 248

degrees() (nipy.algorithms.graph.graph.Graph method), 256

degrees() (nipy.algorithms.graph.graph.WeightedGraph method), 259

demo_ortho_slicer() (in module nipy.labs.viz_tools.slicers), 759

demo_plot_map() (in module nipy.labs.viz_tools.activation_maps), 729

demo_plot_map_3d() (in module nipy.labs.viz_tools.maps_3d), 739

denom_poly() (nipy.algorithms.statistics.rft.ECquasi method), 491

density() (nipy.algorithms.statistics.rft.ChiBarSquared method), 488

density() (nipy.algorithms.statistics.rft.ChiSquared method), 488

density() (nipy.algorithms.statistics.rft.ECcone method), 489

density() (nipy.algorithms.statistics.rft.FStat method), 492

density() (nipy.algorithms.statistics.rft.Hotelling method), 492

density() (nipy.algorithms.statistics.rft.MultilinearForm method), 493

density() (nipy.algorithms.statistics.rft.OneSidedF method), 494

density() (nipy.algorithms.statistics.rft.Roy method), 494

density() (nipy.algorithms.statistics.rft.TStat method), 495

density() (nipy.labs.spatial_models.structural_bfls.LandmarkRegions method), 702

density_per_component() (nipy.algorithms.clustering.von_mises_fisher_mixture.VonMisesMixture method), 216

depth_from_leaves() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 184

depth_from_leaves() (nipy.algorithms.graph.forest.Forest method), 248

depth_from_leaves() (nipy.algorithms.graph.Graph method), 256

depth_from_leaves() (nipy.algorithms.graph.WeightedGraph method), 259

design() (nipy.algorithms.statistics.formula.formulae.Factor method), 355

design() (nipy.algorithms.statistics.formula.formulae.Formula method), 385

design() (nipy.algorithms.statistics.formula.formulae.RandomEffects method), 388

design_expr (nipy.algorithms.statistics.formula.formulae.Factor attribute), 356

design_expr (nipy.algorithms.statistics.formula.formulae.Formula attribute), 386

design_expr (nipy.algorithms.statistics.formula.formulae.RandomEffects attribute), 388

DesignMatrix (class in nipy.modalities.fmri.design_matrix), 775
detsh() (in module nipy.algorithms.clustering.bgmm), 163
deviance() (nipy.algorithms.statistics.models.family.family.Binomial method), 430
deviance() (nipy.algorithms.statistics.models.family.family.Fisher_richlet() (in module nipy.algorithms.clustering.bgmm), 163
deviance() (nipy.algorithms.statistics.models.family.family.Gamma method), 432
deviance() (nipy.algorithms.statistics.models.family.family.Gamma method), 431
deviance() (nipy.algorithms.statistics.models.family.family.Gaussian method), 433
deviance() (nipy.algorithms.statistics.models.family.family.InverseGaussian method), 435
deviance() (nipy.algorithms.statistics.models.family.family.Poisson method), 436
deviance() (nipy.algorithms.statistics.models.family.family.Raoian_right() (in module nipy.modalities.fmri.design_matrix), 776
deviance() (nipy.algorithms.statistics.models.family.family.Raoian_right() (in module nipy.modalities.fmri.design_matrix), 776
devresid() (nipy.algorithms.statistics.models.family.family.Binomial method), 430
devresid() (nipy.algorithms.statistics.models.family.family.Family method), 431
devresid() (nipy.algorithms.statistics.models.family.family.Gamma method), 432
devresid() (nipy.algorithms.statistics.models.family.family.Gaussian method), 434
devresid() (nipy.algorithms.statistics.models.family.family.InverseGaussian method), 435
devresid() (nipy.algorithms.statistics.models.family.family.Poisson method), 436
diff() (nipy.algorithms.statistics.formula.formulae.Beta method), 340
diff() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 371
diff() (nipy.algorithms.statistics.formula.formulae.Term method), 403
diff() (nipy.algorithms.statistics.formula.formulae.Beta method), 340
diff() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 371
diff() (nipy.algorithms.statistics.formula.formulae.Term method), 403
diffusion() (nipy.algorithms.graph.field.Field method), 238
dijkstra() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 184
dijkstra() (nipy.algorithms.graph.field.Field method), 238
dijkstra() (nipy.algorithms.graph.forest.Forest method), 435
dijkstra() (nipy.algorithms.graph.graph.WeightedGraph method), 248
dilation() (nipy.algorithms.graph.field.Field method), 238
dim_cmap() (in module nipy.labs.spatial_models.discrete_domain), 663
dirichlet_eval() (in module nipy.algorithms.clustering.bgmm), 163
DiscreteDomain (class in nipy.labs.spatial_models.discrete_domain), 663
DiscreteParzenMutualInformation (class in nipy.algorithms.registration.similarity_measures), 303
displacement_field (class in nipy.labs.group.displacement_field), 629
method), 749 

draw_samples() (in module nipy.labs.utils.reproducibility_measures), 717 

drop_io_dim() (in module nipy.core.reference.coordinate_map), 552 

DTI, 31 

dtype (nipy.algorithms.statistics.formula.formulae.Factor attribute), 356 

dtype (nipy.algorithms.statistics.formula.formulae.Formula attribute), 386 

dtype (nipy.algorithms.statistics.formula.formulae.RandomTerm attribute), 388 

dtype (nipy.core.reference.coordinate_system.CoordinateSystem attribute), 564 

dummy_eq() (nipy.algorithms.statistics.formula.formulae.Beta method), 340 

dummy_eq() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 371 

dummy_eq() (nipy.algorithms.statistics.formula.formulae.Term method), 403 

dummy_index (nipy.algorithms.statistics.formula.formulae.Beta attribute), 341 

dump() (nipy.labs.statistical_mapping.LinearModel method), 705 

DWI, 31 

ECcone (class in nipy.algorithms.statistics.rft), 489 

ECDF() (in module nipy.algorithms.statistics.models.utils), 484 

ECquasi (class in nipy.algorithms.statistics.rft), 489 

edge_map() (nipy.labs.viz_tools.slicers.BaseSlicer method), 744 

edge_map() (nipy.labs.viz_tools.slicers.BaseStackedSlicer method), 747 

edge_map() (nipy.labs.viz_tools.slicers.OrthoSlicer method), 751 

edge_map() (nipy.labs.viz_tools.slicers.XSlicer method), 753 

edge_map() (nipy.labs.viz_tools.slicers.YSlicer method), 755 

edge_map() (nipy.labs.viz_tools.slicers.ZSlicer method), 758 

EEGlab, 31 

ez() (in module nipy.labs.group.mixed_effects), 633 

enabled (nipy.testing.doctester.NipyDoctest attribute), 837 

enableOpt (nipy.testing.doctester.NipyDoctest attribute), 837 

dp_nn() (in module nipy.algorithms.graph.graph), 263 

equals() (nipy.algorithms.statistics.formula.formulae.Beta method), 341 

equals() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 371 

equals() (nipy.algorithms.statistics.formula.formulae.Term method), 403 

equivalent() (in module nipy.core.reference.coordinate_map), 553 

DWM, 31 

DWM, 31 

dump() (nipy.labs.statistical_mapping.LinearModel method), 705 

Index 889
method), 687

get_inverse() (nipy.labs.datasets.transforms.affine_transform.AffineTransform method), 602

get_inverse() (nipy.labs.datasets.transforms.transform.TransformV() (nipy.algorithms.graph.graph.WeightedGraph method), 256

get_leaves_id() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 677

get_local_maxima() (nipy.algorithms.graph.field.Field method), 239

get_local_volume() (nipy.labs.spatial_models.discrete_domain.NDGridDomain method), 666

get_mask_bounds() (in module nipy.labs.viz_tools.coord_tools), 738

generalize() (nipy.algorithms.statistics.formula.formulae.Beta method), 344

get_vertices() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 185

get_weights() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 185

get_vertices() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 249

get_weights() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 257

get_weights() (nipy.algorithms.graph.field.Field method), 257

get_weights() (nipy.algorithms.graph.forest.Forest method), 260

get_vertices() (nipy.algorithms.graph.forest.Forest method), 240

get_vertices() (nipy.algorithms.graph.graph.Graph method), 257

get_volume() (nipy.labs.spatial_models.discrete_domain.NDGridDomain method), 666

get_volume() (nipy.labs.spatial_models.discrete_domain.StructuredDomain method), 668

get_volume() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 678

get_volume() (nipy.labs.spatial_models.mroi.SubDomains method), 677

get_world_coords() (nipy.labs.datasets.volumes.volume_grid.VolumeGrid method), 619

get_world_coors() (nipy.labs.datasets.volumes.volume_field.VolumeField method), 614

get_world_coords() (nipy.labs.datasets.volumes.volume_field.VolumeField method), 619

get_world_coords() (nipy.labs.datasets.volumes.volume_img.VolumeImg method), 623

get_weights() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 249

get_weights() (nipy.algorithms.graph.graph.WeightedGraph method), 256

get_weights() (nipy.algorithms.graph.graph.WeightedGraph method), 260

get_weights() (nipy.algorithms.graph.graph.WeightedGraph method), 260

get_omega() (nipy.algorithms.statistics.models.nlsmodel.NLSModel method), 458

getparams() (nipy.algorithms.statistics.models.nlsmodel.NLSModel method), 458
is_Equality (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_even (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_even (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_even (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_factor() (in module nipy.algorithms.statistics.formula.formulae), 419
is_factor_term() (in module nipy.algorithms.statistics.formula.formulae), 419
is_finite (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_finite (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_finite (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_Float (nipy.algorithms.statistics.formula.formulae.Beta attribute), 344
is_Float (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_Float (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_function() (in module nipy.algorithms.statistics.formula.formulae), 419
is_hermitian (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_hermitian (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_hermitian (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_image() (in module nipy.core.image.image), 516
is_imaginary (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_imaginary (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_imaginary (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_infinitesimal (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_infinitesimal (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_infinitesimal (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_ Integer (nipy.algorithms.statistics.formula.formulae.Beta attribute), 344
is_integer (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_integer (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_integer (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_irrational (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_irrational (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_irrational (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_Matrix (nipy.algorithms.statistics.formula.formulae.Beta attribute), 344
is_Matrix (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_Matrix (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_noninteger (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_noninteger (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_noninteger (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_nonnegative (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_nonnegative (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_nonnegative (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_negative (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_negative (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_negative (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_nonpositive (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_nonpositive (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_nonpositive (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_positive (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_positive (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_positive (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_real (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_real (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_real (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_realmatrix (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_realmatrix (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_realmatrix (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_squared (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_squared (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_squared (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_unusual (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_unusual (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_unusual (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_zero (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_zero (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_zero (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_nonpositive (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_nonpositive (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_nonpositive (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_nonzero (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_nonzero (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_nonzero (nipy.algorithms.statistics.formula.formulae.Term attribute), 408
is_Not (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_Not (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_Not (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_Number (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_number (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_Number (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_number (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 376
is_Number (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_NumberSymbol (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_NumberSymbol (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_NumberSymbol (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_odd (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_odd (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_odd (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_Order (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_Order (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_Order (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_Piecewise (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_Piecewise (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_Piecewise (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_polar (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_polar (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_polar (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_Poly (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_Poly (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_Poly (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_polynomial() (nipy.algorithms.statistics.formula.formulae.Beta method), 346
is_polynomial() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 376
is_polynomial() (nipy.algorithms.statistics.formula.formulae.Term method), 408
is_positive (nipy.algorithms.statistics.formula.formulae.Beta attribute), 346
is_positive (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 377
is_positive (nipy.algorithms.statistics.formula.formulae.Term attribute), 409
is_Pow (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_Pow (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_Pow (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_prime (nipy.algorithms.statistics.formula.formulae.Beta attribute), 346
is_prime (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 377
is_prime (nipy.algorithms.statistics.formula.formulae.Term attribute), 409
is_Rational (nipy.algorithms.statistics.formula.formulae.Beta attribute), 345
is_rational (nipy.algorithms.statistics.formula.formulae.Beta attribute), 346
is_Rational (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 375
is_rational (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 377
is_Rational (nipy.algorithms.statistics.formula.formulae.Term attribute), 407
is_rational (nipy.algorithms.statistics.formula.formulae.Term attribute), 409
is_rational_function() (nipy.algorithms.statistics.formula.formulae.Beta method), 346
is_rational_function() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 376
is_rational_function() (nipy.algorithms.statistics.formula.formulae.Term method), 407
Neuroimaging in Python Documentation, Release 0.3.0

649 log_gaussian_pdf() (in module nipy.labs.group.spatial_relaxation_onesample), 649
649 log_like() (nipy.algorithms.statistics.mixed_effects_stat.MixedEffectsModel method), 424
649 log_likelihood_ratio() (in module nipy.labs.group.mixed_effects), 633
649 log_weighted_density() (nipy.algorithms.clustering.von_mises_fisher_mixture.VonMisesMixture method), 217
Logit (class in nipy.algorithms.statistics.models.family.links), 442
logL() (nipy.algorithms.statistics.models.glm.Model method), 448
logL() (nipy.algorithms.statistics.models.model.LikelihoodModel method), 452
logL() (nipy.algorithms.statistics.models.model.LikelihoodModelResults static method), 455
logL() (nipy.algorithms.statistics.models.regression.ARModel method), 465
logL() (nipy.algorithms.statistics.models.regression.GLModel method), 467
logL() (nipy.algorithms.statistics.models.regression.OLSModel method), 470
logL() (nipy.algorithms.statistics.models.regression.RegressionResults static method), 474
logL() (nipy.algorithms.statistics.models.regression.WLSModel method), 476
loss() (nipy.algorithms.registration.similarity_measures.CorrelationCoefficient method), 302
loss() (nipy.algorithms.registration.similarity_measures.CorrelationRatio method), 302
loss() (nipy.algorithms.registration.similarity_measures.DiscreteParzenMutualInformation method), 303
loss() (nipy.algorithms.registration.similarity_measures.DiscreteParzenMutualInformation method), 303
loss() (nipy.algorithms.registration.similarity_measures.MutualInformation method), 304
loss() (nipy.algorithms.registration.similarity_measures.NormalizedMutualInformation method), 304
loss() (nipy.algorithms.registration.similarity_measures.ParzenMutualInformation method), 305
loss() (nipy.algorithms.registration.similarity_measures.SimilarityMeasure method), 305
loss() (nipy.algorithms.registration.similarity_measures.SupervisedLikelihoodRatio method), 305
ls series() (nipy.algorithms.statistics.formula.formulae.Beta method), 348
ls series() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 378
ls series() (nipy.algorithms.statistics.formula.formulae.Term method), 411
M
m2screenshot() (in module nipy.labs.viz_tools.maps_3d), 739
mad() (in module nipy.algorithms.statistics.models.utils), 739
main() (in module nipy.algorithms.clustering.imm), 212
main() (in module nipy.utils.perlpie), 854
main_cc() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 240
main_cc() (nipy.algorithms.graph.field.Field method), 250
main_cc() (nipy.algorithms.graph.graph.Graph method), 257
main_cc() (nipy.algorithms.graph.graph.WeightedGraph method), 260
main_effect (nipy.algorithms.statistics.formula.formulae.Factor attribute), 356
make_affine() (nipy.core.reference.coordinate_map.CoordMapMaker method), 545
make_bsa_image() (in module nipy.labs.spatial_models.bsa_io), 661
make_cmap() (nipy.core.reference.coordinate_map.CoordMapMaker method), 546
make_data() (in module nipy.labs.glm.benchmarks.bench_glm), 625
make_dmtx() (in module nipy.modalities.fmri.design_matrix), 776
make_dummy() (in module nipy.algorithms.statistics.formula.formulae), 419
make_feature() (nipy.labs.spatial_models.parcellation.MultiSubjectParcellation method), 698
make_feature() (nipy.labs.spatial_models.discrete_domain.NDGridDomain method), 667
make_forest() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 670
make_graph() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 678
make_grid() (in module nipy.algorithms.registration.groupwise_registration), 288
make_grid() (in module nipy.algorithms.clustering.gmm.GridDescriptor method), 178
make_hroi_from_subdomain() (in module nipy.labs.spatial_models.hroi), 683
make_job() (in module nipy.interfaces.spm), 586
make_label_dec() (in module nipy.testing.decorators), 833
make_recarray() (in module nipy.algorithms.statistics.formula.formulae), 419

Index 903
nipy.labs.utils.reproducibility_measures, 720
MixedEffectsModel (class in nipy.algorithms.statistics.mixed_effects_stat), 423
MixedIMM (class in nipy.algorithms.clustering.imm), 203
mixture_density() (nipy.algorithms.clustering.von_mises_fisher_mixture method), 217
mixture_likelihood() (nipy.algorithms.clustering.bgmm.BGMM method), 153
mixture_likelihood() (nipy.algorithms.clustering.bgmm.VBGMM method), 160
mixture_likelihood() (nipy.algorithms.clustering.imm.IMM method), 199
mixture_likelihood() (nipy.algorithms.clustering.imm.MixedIMM method), 207
mlab_tempfile() (in module nipy.interfaces.matlab), 583
Model (class in nipy.algorithms.statistics.models.glm), 447
Model (class in nipy.algorithms.statistics.models.model), 456
model_generator() (in module nipy.modalities.fmri.fmristat.model), 795
ModelOutputImage (class in nipy.modalities.fmri.fmristat.model), 794
moment_matching() (in module nipy.algorithms.segmentation.segmentation), 314
monotone_fn_inverter() (in module nipy.algorithms.statistics.formula.formulae), 484
MSE() (nipy.algorithms.statistics.models.regression.RegressionResults static method), 473
MSR() (nipy.algorithms.statistics.models.regression.RegressionResults static method), 473
mst() (in module nipy.algorithms.graph.graph), 264
MST() (nipy.algorithms.statistics.models.regression.RegressionResults static method), 473
Mstep() (nipy.algorithms.clustering.ggmixture.GGGM method), 168
Mstep() (nipy.algorithms.clustering.ggmixture.GGM method), 171
mu_ball() (in module nipy.algorithms.statistics.rft), 496
mu_sphere() (in module nipy.algorithms.statistics.rft), 496
MultilinearForm (class in nipy.algorithms.statistics.rft), 493
multinomial() (in module nipy.algorithms.clustering.bgmm), 164
multiple_fast_inv() (in module nipy.algorithms.statistics.utils), 500
multiple_mahalanobis() (in module nipy.algorithms.clustering.ggmixture), 176

MultiSubjectParcellation (class in nipy.labs.spatial_models.parcellation), 697
multivariate_stat (class in nipy.labs.group.spatial_relaxation_onesample), 646
MutualInformation (class in nipy.algorithms.registration.similarity_measures), 303

N
n() (nipy.algorithms.statistics.formula.formulae.Beta method), 349
n() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 379
n() (nipy.algorithms.statistics.formula.formulae.Term method), 411
name (nipy.algorithms.statistics.formula.formulae.Beta attribute), 349
name (nipy.algorithms.statistics.formula.formulae.FactorTerm attribute), 379
name (nipy.algorithms.statistics.formula.formulae.Term attribute), 412
name (nipy.core.reference.coordinate_system.CoordinateSystem attribute), 564
name (nipy.testing.doctester.NipyDoctest attribute), 838
natural_spline() (in module nipy.algorithms.statistics.models.formulae), 420
natural_spline() (in module nipy.modalities.fmri.design), 772
NDGridDomain (class in nipy.labs.spatial_models.discrete_domain), 665

ndim (nipy.core.reference.coordinate_system.CoordinateSystem attribute), 564
ndim() (nipy.core.image.image.Image static method), 513
ndims (nipy.core.reference.coordinate_map.AffineTransform attribute), 542

ndims (nipy.core.reference.coordinate_map.CoordinateMap attribute), 549
needs_review() (in module nipy.testing.decorators), 834
next() (nipy.algorithms.statistics.models.glm.Model method), 449
next() (nipy.algorithms.statistics.models.nlsmodel.NLSModel method), 458
next() (nipy.core.image.image_list.ImageList method), 525
next() (nipy.modalities.fmri.fmri.FmriImageList method), 785
nifti2nipy() (in module nipy.io.nifti_ref), 593
NiftiError (class in nipy.io.nifti_ref), 593
nipy.algorithms.clustering.bgmm (module), 149
nipy.algorithms.clustering.ggmixture (module), 167
nipy.algorithms.clustering.gmm (module), 173
nipy.algorithms.clustering.hierarchical_clustering (module), 181
nipy.algorithms.clustering.imm (module), 195
nipy.algorithms.clustering.utils (module), 213
nipy.algorithms.clustering.von_mises_fisher_mixture (module), 215
nipy.algorithms.diagnostics.screens (module), 221
nipy.algorithms.diagnostics.timediff (module), 223
nipy.algorithms.diagnostics.tsdiffplot (module), 225
nipy.algorithms.fwhm (module), 227
nipy.algorithms.graph.bipartite_graph (module), 231
nipy.algorithms.graph.field (module), 235
nipy.algorithms.graph.forest (module), 245
nipy.algorithms.interpolation (module), 265
nipy.algorithms.kernel_smooth (module), 267
nipy.algorithms.optimize (module), 271
nipy.algorithms.registration.affine (module), 273
nipy.algorithms.registration.chain_transform (module), 281
nipy.algorithms.registration.groupwise_registration (module), 283
nipy.algorithms.registration.histogram_registration (module), 291
nipy.algorithms.registration.optimizer (module), 295
nipy.algorithms.registration.polyaffine (module), 297
nipy.algorithms.registration.resample (module), 299
nipy.algorithms.registration.similarity_measures (module), 301
nipy.algorithms.registration.transform (module), 307
nipy.algorithms.resample (module), 309
nipy.algorithms.segmentation.brain_segmentation (module), 311
nipy.algorithms.segmentation.segmentation (module), 313
nipy.algorithms.statistics.bench.bench_intvol (module), 319
nipy.algorithms.statistics.empirical_pvalue (module), 319
nipy.algorithms.statistics.formula.formulae (module), 325
nipy.algorithms.statistics.mixed_effects_stat (module), 423
nipy.algorithms.statistics.models.family.family (module), 429
nipy.algorithms.statistics.models.family.links (module), 439
nipy.algorithms.statistics.models.family.varfuncs (module), 445
nipy.algorithms.statistics.models.glm (module), 447
nipy.algorithms.statistics.models.model (module), 451
nipy.algorithms.statistics.models.nlsmodel (module), 457
nipy.algorithms.statistics.models.regression (module), 461
nipy.algorithms.statistics.models.setupscons (module), 481
nipy.algorithms.statistics.models.utils (module), 483
nipy.algorithms.statistics.onesample (module), 485
nipy.algorithms.statistics.rft (module), 487
nipy.algorithms.statistics.utils (module), 499
nipy.algorithms.utils.fast_distance (module), 501
nipy.algorithms.utils.matrices (module), 503
nipy.algorithms.utils.pca (module), 507
nipy.core.image.image (module), 511
nipy.core.image.image_list (module), 523
nipy.core.image.image_spaces (module), 527
nipy.core.reference.array_coords (module), 533
nipy.core.reference.coordinate_map (module), 537
nipy.core.reference.coordinate_system (module), 561
nipy.core.reference.slices (module), 567
nipy.core.reference.spaces (module), 571
nipy.core.utils.matlab (module), 579
nipy.interfaces.matlab (module), 583
nipy.interfaces.spm (module), 585
nipy.io.files (module), 587
nipy.io.nifti_ref (module), 591
nipy.labs.bindings.benchmarks.bench_numpy (module), 597
nipy.labs.datasets.converters (module), 599
nipy.labs.datasets.transforms.affine_transform (module), 601
nipy.labs.datasets.transforms.affine Utils (module), 603
nipy.labs.datasets.transforms.transform (module), 605
nipy.labs.datasets.volumes.volume_data (module), 609
nipy.labs.datasets.volumes.volume_field (module), 613
nipy.labs.datasets.volumes.volume Grid (module), 617
nipy.labs.datasets.volumes.volume_img (module), 621
nipy.labs.glm.benchmarks.bench_glm (module), 625
nipy.labs.glm.glm (module), 627
nipy.labs.group.displacement_field (module), 629
nipy.labs.group.mixed_effects (module), 633
nipy.labs.group.permutation_test (module), 635
nipy.labs.group.spatial_relaxation_onesample (module), 645
nipy.labs.mask (module), 651
nipy.labs.spatial_models.bayesian_structural_analysis (module), 655
nipy.labs.spatial_models.bsa_iO (module), 661
nipy.labs.spatial_models.discrete_domain (module), 663
nipy.labs.spatial_models.hierarchical_parcellation (module), 673
nipy.labs.spatial_models.hroi (module), 675
nipy.labs.spatial_models.mroi (module), 685
nipy.labs.spatial_models.parcel_io (module), 693
nipy.labs.spatial_models.parcellation (module), 697
nipy.labs.spatial_models.structural_bfls (module), 701
nipy.labs.statistical_mapping (module), 705
nipy.labs.utils.random_threshold (module), 709
nonzero() (in module nipy.algorithms.registration.similarity_measures), 306
nonzero() (in module nipy.labs.group.mixed_effects), 633
norm_resid() (nipy.algorithms.statistics.models.regression.RegressionResults
static method), 474
normal() (nipy.algorithms.statistics.formula.formulae.Beta
method), 349
normal() (nipy.algorithms.statistics.formula.formulae.FactorTerm
method), 379
normal() (nipy.algorithms.statistics.formula.formulae.Term
method), 412
normal_eval() (in module nipy.algorithms.clustering.bgmm), 165
NormalEmpiricalNull (class in nipy.algorithms.statistics.empirical_pvalue),
42, 319
normalization (nipy.algorithms.kernel_smooth.LinearFilter
attribute), 268
normalize() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest
method), 186
normalize() (nipy.algorithms.graph.field.Field method),
240
normalize() (nipy.algorithms.graph.forest.Forest method),
250
normalize() (nipy.algorithms.graph.graph.WeightedGraph
method), 260
normalized_external_field() (nipy.algorithms.segmentation.segmentation.Segmentation
method), 314
NormalizedMutualInformation (class in nipy.algorithms.registration.similarity_measures),
304
npoints() (nipy.algorithms.registration.similarity_measures.CorrelationCoefficient
method), 302
npoints() (nipy.algorithms.registration.similarity_measures.CorrelationRatio
method), 302
npoints() (nipy.algorithms.registration.similarity_measures.CorrelationRatioL1
method), 303
npoints() (nipy.algorithms.registration.similarity_measures.DiscreteParzenMutualInformation
method), 303
npoints() (nipy.algorithms.registration.similarity_measures.NormalizedMutualInformation
method), 303
npoints() (nipy.algorithms.registration.similarity_measures.MutualInformation
method), 303
npoints() (nipy.algorithms.registration.similarity_measures.ParzenMutualInformation
method), 303
npoints() (nipy.algorithms.registration.similarity_measures.SupervisedLikelihoodRatio
method), 303
npoints() (nipy.algorithms.registration.similarity_measures.DiscreteParzenMutualInformation
method), 303
npoints() (nipy.algorithms.registration.similarity_measures.MutualInformation
method), 303
npoints() (nipy.algorithms.registration.similarity_measures.ParzenMutualInformation
method), 303
npoints() (nipy.algorithms.registration.similarity_measures.SupervisedLikelihoodRatio
method), 303
nseries() (nipy.algorithms.statistics.formula.formulae.Beta
method), 349
nseries() (nipy.algorithms.statistics.formula.formulae.FactorTerm
method), 379
NLSModel (class in nipy.algorithms.statistics.models.nlsmodel),
457
Index 907
<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nseries()</td>
<td>(nipy.algorithms.statistics.formula.formulae.Term method), 412</td>
</tr>
<tr>
<td>nsimplify()</td>
<td>(nipy.algorithms.statistics.formula.formulae.Beta method), 349</td>
</tr>
<tr>
<td>nsimplify()</td>
<td>(nipy.algorithms.statistics.formula.formulae.FactorTerm method), 379</td>
</tr>
<tr>
<td>nsimplify()</td>
<td>(nipy.algorithms.statistics.formula.formulaeTERM method), 412</td>
</tr>
<tr>
<td>OLS</td>
<td>(class in nipy.modalities.fmri.fmristat.model), 794</td>
</tr>
<tr>
<td>ols()</td>
<td>(in module nipy.labs.glm.benchmarks.bench_glm), 625</td>
</tr>
<tr>
<td>ols()</td>
<td>(module nipy.labs.glm), 628</td>
</tr>
<tr>
<td>OLSModel</td>
<td>(class in nipy.algorithms.statistics.models.regression), 469</td>
</tr>
<tr>
<td>one_sample_ftest()</td>
<td>(in module nipy.algorithms.statistics.mixed_effects_stat), 426</td>
</tr>
<tr>
<td>one_sample_ttest()</td>
<td>(in module nipy.algorithms.statistics.mixed_effects_stat), 426</td>
</tr>
<tr>
<td>onesample_stat()</td>
<td>(in module nipy.labs.group.permutation_test), 642</td>
</tr>
<tr>
<td>onesample_test()</td>
<td>(in module nipy.labs.statistical_mapping), 706</td>
</tr>
<tr>
<td>OneSidedF</td>
<td>(class in nipy.algorithms.statistics.rft), 494</td>
</tr>
<tr>
<td>open4csv()</td>
<td>(in module nipy.utils.compat3), 851</td>
</tr>
<tr>
<td>opening()</td>
<td>(nipy.algorithms.graph.field.Field method), 241</td>
</tr>
<tr>
<td>optimize()</td>
<td>(nipy.algorithms.registration.histogram_registration.HistogramRegistration method), 292</td>
</tr>
<tr>
<td>options()</td>
<td>(nipy.testing.doctester.NipyDoctest method), 838</td>
</tr>
<tr>
<td>order</td>
<td>(nipy.algorithms.statistics.rft.ECQuasi attribute), 492</td>
</tr>
<tr>
<td>orth()</td>
<td>(in module nipy.modalities.fmri.spm.reml), 819</td>
</tr>
<tr>
<td>orth_axes()</td>
<td>(in module nipy.core.reference.coordinate_map), 555</td>
</tr>
<tr>
<td>OrthoSlicer</td>
<td>(class in nipy.labs.viz_tools.slicers), 749</td>
</tr>
<tr>
<td>output_AR1()</td>
<td>(in module nipy.modalities.fmri.fmristat.model), 796</td>
</tr>
<tr>
<td>output_AR1()</td>
<td>(in module nipy.modalities.fmri.fmristat.outputters), 801</td>
</tr>
<tr>
<td>output_difference()</td>
<td>(nipy.testing.doctester.NipyOutputChecker method), 839</td>
</tr>
<tr>
<td>output_F()</td>
<td>(in module nipy.modalities.fmri.fmristat.model), 796</td>
</tr>
<tr>
<td>output_F()</td>
<td>(in module nipy.modalities.fmri.fmristat.outputters), 801</td>
</tr>
<tr>
<td>output_space</td>
<td>(nipy.labs.datasets.transforms.affine_transform.AffineTransform attribute), 602</td>
</tr>
<tr>
<td>output_T()</td>
<td>(in module nipy.modalities.fmri.fmristat.model), 796</td>
</tr>
<tr>
<td>output_T()</td>
<td>(in module nipy.modalities.fmri.fmristat.outputters), 801</td>
</tr>
<tr>
<td>overall_loss()</td>
<td>(nipy.algorithms.registration.similarity_measures.CorrelationCoefficient method), 302</td>
</tr>
<tr>
<td>overall_loss()</td>
<td>(nipy.algorithms.registration.similarity_measures.CorrelationRatio method), 302</td>
</tr>
<tr>
<td>overall_loss()</td>
<td>(nipy.algorithms.registration.similarity_measures.CorrelationRatioL1 method), 303</td>
</tr>
<tr>
<td>overall_loss()</td>
<td>(nipy.algorithms.registration.similarity_measures.DiscreteParzenMutualInformation method), 303</td>
</tr>
<tr>
<td>overall_loss()</td>
<td>(nipy.algorithms.registration.similarity_measures.MutualInformation method), 304</td>
</tr>
<tr>
<td>overall_loss()</td>
<td>(nipy.algorithms.registration.similarity_measures.NormalizedMutualInformation method), 304</td>
</tr>
<tr>
<td>overall_loss()</td>
<td>(nipy.algorithms.registration.similarity_measures.ParzenMutualInformation method), 305</td>
</tr>
<tr>
<td>overall_loss()</td>
<td>(nipy.algorithms.registration.similarity_measures.SupervisedLikelihoodRatio method), 306</td>
</tr>
<tr>
<td>p_value()</td>
<td>(nipy.modalities.fmri.glm.Contrast method), 804</td>
</tr>
<tr>
<td>Paradigm</td>
<td>(class in nipy.modalities.fmri.experimental_paradigm), 780</td>
</tr>
<tr>
<td>param</td>
<td>(nipy.algorithms.registration.affine.Affine attribute), 274</td>
</tr>
<tr>
<td>param</td>
<td>(nipy.algorithms.registration.affine.Affine2D attribute), 275</td>
</tr>
<tr>
<td>param</td>
<td>(nipy.algorithms.registration.affine.Rigid attribute), 276</td>
</tr>
<tr>
<td>param</td>
<td>(nipy.algorithms.registration.affine.Rigid2D attribute), 277</td>
</tr>
<tr>
<td>param</td>
<td>(nipy.algorithms.registration.affine.Similarity attribute), 278</td>
</tr>
<tr>
<td>param</td>
<td>(nipy.algorithms.registration.affine.Similarity2D attribute), 279</td>
</tr>
<tr>
<td>param</td>
<td>(nipy.algorithms.registration.chain_transform.ChainTransform attribute), 282</td>
</tr>
<tr>
<td>param</td>
<td>(nipy.algorithms.registration.polyaffine.PolyAffine attribute), 298</td>
</tr>
</tbody>
</table>
rank() (nipy.algorithms.statistics.models.regression.OLSModel static method), 471
rank() (nipy.algorithms.statistics.models.regression.WLSModel static method), 477
ratsimp() (nipy.algorithms.statistics.formula.formulae.Beta method), 350
ratsimp() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 380
ratsimp() (nipy.algorithms.statistics.formula.formulae.Term method), 412
Realign4d (class in nipy.algorithms.registration.groupwise_registration), 285
realign4d() (in module nipy.algorithms.registration.groupwise_registration), 288
Realign4dAlgorithm (class in nipy.algorithms.registration.groupwise_registration), 287
recipr0() (in module nipy.algorithms.utils.matrices), 504
recompute_labels() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 679
recompute_labels() (nipy.labs.spatial_models.mroi.SubDomains method), 688
reduce() (nipy.algorithms.clustering.imm.IMM method), 200
reduce() (nipy.algorithms.clustering.imm.MixedIMM method), 208
reduce_coo_matrix() (in module nipy.labs.spatial_models.discrete_domain), 671
reduce_to_leaves() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 679
reference() (nipy.core.image.image.Image static method), 513
refine() (nipy.algorithms.statistics.formula.formulae.Beta method), 350
refine() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 380
refine() (nipy.algorithms.statistics.formula.formulae.Term method), 412
register_to() (nipy.core.reference.spaces.XYZSpace method), 573
RegressionOutput (class in nipy.modalities.fmri.fmristat.outputters), 799
RegressionOutputList (class in nipy.modalities.fmri.fmristat.outputters), 800
RegressionResults (class in nipy.algorithms.statistics.models.regression), 471
reml() (in module nipy.modalities.fmri.spm.reml), 819
remove_edges() (nipy.algorithms.graph.forest.Forest method), 251
remove_edges() (nipy.algorithms.graph.field.Field method), 241
remove_edges() (nipy.algorithms.graph.forest.Forest method), 251
remove_edges() (nipy.algorithms.graph.field.Field method), 241
remove_trivial_edges() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 187
remove_trivial_edges() (nipy.algorithms.graph.field.Field method), 241
remove_trivial_edges() (nipy.algorithms.graph.forest.Forest method), 251
remove_trivial_edges() (nipy.algorithms.graph.field.Field method), 241
removeO() (nipy.algorithms.statistics.formula.formulae.Beta method), 350
removeO() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 380
removeO() (nipy.algorithms.statistics.formula.formulae.Term method), 412
renamed_axes() (nipy.core.image.image.Image method), 513
renamed_domain() (in module nipy.core.reference.coordinate_map), 557
renamed_domain() (nipy.core.reference.coordinate_map.AffineTransform method), 542
renamed_domain() (nipy.core.reference.coordinate_map.CoordinateMap method), 549
renamed_range() (in module nipy.core.reference.coordinate_map), 557
renamed_range() (nipy.core.reference.coordinate_map.AffineTransform method), 543
renamed_range() (nipy.core.reference.coordinate_map.CoordinateMap method), 549
reorder_from_leaves_to_roots() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 187
reorder_from_leaves_to_roots() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 187
reordered_axes() (nipy.core.image.image.Image method), 514
reordered_domain() (nipy.core.reference.coordinate_map.AffineTransform method), 543
reordered_domain() (nipy.core.reference.coordinate_map.CoordinateMap method), 550
reordered_domain() (nipy.core.reference.coordinate_map.AffineTransform method), 543
reordered_domain() (nipy.core.reference.coordinate_map.CoordinateMap method), 550
reordered_range() (in module nipy.core.reference.coordinate_map), 558
reordered_range() (nipy.core.reference.coordinate_map.AffineTransform method), 543
reordered_range() (nipy.core.reference.coordinate_map.CoordinateMap method), 550
reordered_reference() (nipy.core.image.image.Image method), 515
replace() (nipy.algorithms.statistics.formula.formulae.Beta method), 350
replace() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 380
replace() (nipy.algorithms.statistics.formula.formulae.Term method), 412
replace_inside() (in module nipy.labs.viz_tools.cm), 735
representative_feature() (nipy.labs.spatial_models.discrete_domain.DiscreteDomain method), 664
representative_feature() (nipy.labs.spatial_models.discrete_domain.NDGridDomain method), 667
representative_feature() (nipy.labs.spatial_models.discrete_domain.NDGridDomain method), 669
representative_feature() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 680
representative_feature() (nipy.labs.spatial_models.mroi.SubDomains method), 689
resample() (in module nipy.algorithms.registration.resample), 299
resample() (in module nipy.algorithms.registration.resample), 309
resample() (nipy.algorithms.registration.groupwise_registration.Resample4d method), 284
resample() (nipy.algorithms.registration.groupwise_registration.Resample4d method), 286
resample() (nipy.algorithms.registration.groupwise_registration.Resample4d method), 288
resample4d() (in module nipy.algorithms.registration.groupwise_registration), 289
resample_full_data() (nipy.algorithms.registration.groupwise_registration.Resample4d method), 288
resample_img2img() (in module nipy.algorithms.registration.resample), 309
resample_regressor() (in module nipy.modalities.fmri.hemodynamic_models), 810
resampled_to_img() (nipy.labs.datasets.volumes.volume_data.VolumeData method), 611
resampled_to_img() (nipy.labs.datasets.volumes.volume_field.VolumeField method), 60, 614
resampled_to_img() (nipy.labs.datasets.volumes.volume_grid.VolumeGrid method), 619
resampled_to_img() (nipy.labs.datasets.volumes.volume_img.VolumeImg method), 623
resel2fwhm() (nipy.algorithms.fwhm.ReselImage method), 228
resel2fwhm() (nipy.algorithms.fwhm.Resels method), 229
ReselImage (class in nipy.algorithms.fwhm), 227
Resels (class in nipy.algorithms.fwhm), 228
results_generator() (nipy.modalities.fmri.fmristat.model), 797
right_incidence() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 217
right_incidence() (nipy.algorithms.graph.field.Field method), 241
right_incidence() (nipy.algorithms.graph.forest.Forest method), 251
right_incidence() (nipy.algorithms.graph.graph.WeightedGraph method), 261
Rigid (class in nipy.algorithms.registration.affine), 275
Rigid2D (class in nipy.algorithms.registration.affine), 277
roi_confidence() (nipy.labs.spatial_models.structural_bfls.LandmarkRegions method), 703
to_fwhm() (nipy.labs.datasets.volumes.structural_bfls.LandmarkRegions method), 703
to_Roi() (nipy.labs.datasets.volumes.volume_data.VolumeData method), 517
rolling() (in module nipy.core.image.image), 518
to_Rigid() (nipy.algorithms.registration.affine.Affine attribute), 274
rotation (nipy.algorithms.registration.affine.Affine2D attribute), 275
rotation (nipy.algorithms.registration.affine.Rigid attribute), 277
rotation (nipy.algorithms.registration.affine.Similarity attribute), 278
rotation (nipy.algorithms.registration.affine.Similarity2D attribute), 279
rotation_mat2vec() (in module nipy.algorithms.registration.affine), 279
rotation_vec2mat() (in module nipy.algorithms.registration.affine), 279
round() (nipy.algorithms.statistics.formula.formulae.Beta method), 351
Index 913
round() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 381
round() (nipy.algorithms.statistics.formula.formulae.Term method), 414
round_numbers() (in module nipy.testing.doctester), 840
Roy (class in nipy.algorithms.statistics.rft), 494
run() (nipy.algorithms.segmentation.segmentation.Segmentation method), 314
run_jobdef() (in module nipy.interfaces.spm), 586
run_matlab() (in module nipy.interfaces.matlab), 583
run_matlab_script() (in module nipy.interfaces.matlab), 583
safe_dtype() (in module nipy.core.reference.coordinate_system), 566
sample() (nipy.algorithms.clustering.bgmm.BGMM method), 153
sample() (nipy.algorithms.clustering.bgmm.VBGMM method), 160
sample() (nipy.algorithms.clustering.imm.IMM method), 200
sample() (nipy.algorithms.clustering.imm.MixedIMM method), 208
sample() (nipy.algorithms.clustering.bgmm.BGMM method), 208
sample() (nipy.algorithms.clustering.bgmm.VBGMM method), 208
sample() (nipy.algorithms.clustering.imm.IMM method), 208
sample() (nipy.algorithms.clustering.imm.MixedIMM method), 208
sample_condition() (in module nipy.modalities.fmri.hemodynamic_models), 811
sample_indicator() (nipy.algorithms.clustering.bgmm.BGMM method), 154
sample_indicator() (nipy.algorithms.clustering.bgmm.VBGMM method), 161
sample_indicator() (nipy.algorithms.clustering.imm.IMM method), 200
sample_indicator() (nipy.algorithms.clustering.imm.MixedIMM method), 209
sample_log_conditional_posterior() (nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method), 648
score() (nipy.algorithms.statistics.models.glm.Model method), 449
score() (nipy.algorithms.statistics.models.models.regression.ARModel method), 466
score() (nipy.algorithms.statistics.models.models.regression.OLSModel method), 471
score() (nipy.algorithms.statistics.models.models.regression.WLSModel method), 477
score() (nipy.algorithms.statistics.models.models.regression.GLSModel method), 468
score() (nipy.algorithms.statistics.models.models.regression.OLSModel method), 471
score() (nipy.algorithms.statistics.models.models.regression.WLSModel method), 477
screen() (in module nipy.algorithms.diagnostics.screens), 221
SecondStage (class in nipy.modalities.fmri.spm.model), 817
SecondStage (class in nipy.algorithms.segmentation.segmentation), 313
select_id() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 680
select_id() (nipy.labs.spatial_models.mroi.SubDomains method), 689
select_roi() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 680
select_roi() (nipy.labs.spatial_models.mroi.SubDomains method), 689
save() (in module nipy.io.files), 588
save() (in module nipy.labs.datasets.converters), 599
select_vmm() (in module nipy.algorithms.clustering.von_mises_fisher_mixture), 218
select_vmm_cv() (in module nipy.algorithms.clustering.von_mises_fisher_mixture), 219

separate() (nipy.algorithms.statistics.formula.formulae.Beta method), 352
separate() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 382
separate() (nipy.algorithms.statistics.formula.formulae.Term method), 414
series() (nipy.algorithms.statistics.formula.formulae.Beta method), 352
series() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 382
series() (nipy.algorithms.statistics.formula.formulae.Term method), 414
series_from_mask() (in module nipy.labs.mask), 654
set() (nipy.algorithms.clustering.gmm.GridDescriptor method), 178
set_constant_densities() (nipy.algorithms.clustering.imm.IMM method), 201
set_constant_densities() (nipy.algorithms.clustering.imm.MixedIMM method), 209
set_edges() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 187
set_edges() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 188
set_edges() (nipy.algorithms.graph.field.Field method), 241
set_edges() (nipy.algorithms.graph.field.Field method), 241
set_edges() (nipy.algorithms.graph.forest.Forest method), 251
set_edges() (nipy.algorithms.graph.forest.FirmForest method), 295
set_edges() (nipy.algorithms.graph.forest.RandomForest method), 257
set_edges() (nipy.algorithms.graph.forest.RandomForest method), 257
set_edges() (nipy.algorithms.graph.forest.RandomForest method), 257
set_edges() (nipy.algorithms.graph.forest.RandomForest method), 257
set_edges() (nipy.algorithms.graph.graph.WeightedGraph method), 261
set_edges() (nipy.labs.spatial_models.discrete_domain.DiscreteDomain method), 665
set_edges() (nipy.labs.spatial_models.discrete_domain.DiscreteDomain method), 667
set_edges() (nipy.labs.spatial_models.discrete_domain.StructuredDomain method), 669
set_edges() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 680
set_edge() (nipy.labs.spatial_models.hroi.HierarchicalROI method), 680
set_feature() (nipy.labs.spatial_models.parcellation.MultiSubjectParcellation method), 698
set_time() (nipy.labs.spatial_models.parcellation.MultiSubjectParcellation method), 698
set_markov_prior() (nipy.algorithms.segmentation.segmentation.Segmentation method), 314
set_priors() (nipy.algorithms.clustering.bgmm.BGMM method), 154
set_priors() (nipy.algorithms.clustering.bgmm.VBGMM method), 161
set_priors() (nipy.algorithms.clustering.imm.IMM method), 201
set_priors() (nipy.algorithms.clustering.imm.MixedIMM method), 209
set_priors() (nipy.labs.spatial_models.parcellation.MultiSubjectParcellation method), 699
set_priors() (nipy.labs.spatial_models.parcellation.MultiSubjectParcellation method), 699
set_priors() (nipy.labs.spatial_models.parcellation.MultiSubjectParcellation method), 699

Index 915
shape_generator() (in module nipy.core.utils.generators), 580
shifted_domain_origin() (in module nipy.core.reference.coordinate_map), 559
shifted_range_origin() (in module nipy.core.reference.coordinate_map), 559
show() (nipy.algorithms.clustering.bgmm.BGMM method), 154
show() (nipy.algorithms.clustering.bgmm.VBGMM method), 161
show() (nipy.algorithms.clustering.ggmmixture.GGM method), 170
show() (nipy.algorithms.clustering.ggmmixture.GGM method), 171
show() (nipy.algorithms.clustering.imm.IMM method), 201
show() (nipy.algorithms.clustering.imm.MixedIMM method), 209
show() (nipy.algorithms.clustering.von_mises_fisher_mixture.VonMisesMixture method), 217
show() (nipy.algorithms.graph.field.Field method), 242
show() (nipy.algorithms.graph.forest.Forest method), 252
show() (nipy.algorithms.graph.Graph method), 257
show() (nipy.algorithms.graph.Graph method), 261
show() (nipy.labs.spatial_models.structural_bfls.LandmarkRegions method), 703
show() (nipy.modalities.fmri.design_matrix.DesignMatrix method), 775
show_components() (nipy.algorithms.clustering.bgmm.BGMM method), 155
show_components() (nipy.algorithms.clustering.bgmm.VBGMM method), 161
show_components() (nipy.algorithms.clustering.imm.IMM method), 201
show_components() (nipy.algorithms.clustering.imm.MixedIMM method), 209
sigma2fwhm() (in module nipy.algorithms.kernel_smooth), 268
signal_to_posterior() (in module nipy.labs.spatial_models.bayesianstructural_analysis), 660
similar_to() (nipy.core.reference.coordinate_map.AffineTransform method), 544
similar_to() (nipy.core.reference.coordinate_map.CoordinateMap method), 550
similar_to() (nipy.core.reference.coordinate_system.CoordinateSystem method), 23
sigma2fwhm() (in module nipy.algorithms.kernel_smooth.LinearFilter method), 268
signal_to_posterior() (in module nipy.labs.spatial_models.bayesianstructural_analysis), 660
similar_to() (nipy.core.reference.coordinate_map.AffineTransform method), 544
similar_to() (nipy.core.reference.coordinate_map.CoordinateMap method), 550
similar_to() (nipy.core.reference.coordinate_system.CoordinateSystem method), 23
Similarity (class in nipy.algorithms.registration.affine), 277
similarity (nipy.algorithms.registration.histogramregistration.HistogramRegistration method), 293
Similarity2D (class in nipy.algorithms.registration.affine), 278
SimilarityMeasure (class in nipy.algorithms.registration.similarity_measures), 305
simple_update() (nipy.algorithms.clustering.imm.IMM method), 201
simple_update() (nipy.algorithms.clustering.imm.MixedIMM method), 210
simplify() (nipy.algorithms.statistics.formula.formulae.Beta method), 353
simplify() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 383
simplify() (nipy.algorithms.statistics.formula.formulae.Term method), 416
simulated_pvalue() (in module nipy.labs.statistical_mapping), 707
skip_doctest_if() (in module nipy.testing.decorators), 834
skip_if_running_nose() (in module nipy.utils.skip_test), 855
slice_generator() (in module nipy.core.utils.generators), 580
slice2aff() (in module nipy.algorithms.registration.affine), 280
smatrix_from_3d_array() (in module nipy.algorithms.registration.histogramregistration), 294
smatrix_from_3d_idx() (in module nipy.labs.spatial_models.discrete_domain), 671
smatrix_from_3d_array() (in module nipy.labs.spatial_models.discrete_domain), 671
smatrix_from_3d_idx() (in module nipy.labs.spatial_models.discrete_domain), 671
smatrix_from_nd_array() (in module nipy.labs.spatial_models.discrete_domain), 671
smatrix_from_nd_idx() (in module nipy.labs.spatial_models.discrete_domain), 671
smooth() (nipy.algorithms.kernel_smooth.LinearFilter method), 268
smoothed_histogram_from_samples() (in module nipy.algorithms.statistics.empirical_pvalue), 323
sort_key() (nipy.algorithms.statistics.formula.formulae.Beta method), 353
sort_key() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 383
sort_key() (nipy.algorithms.statistics.formula.formulae.Term method), 416
sorted_values() (in module nipy.labs.group.permutation_test), 642
SpaceError (class in nipy.core.reference.spaces), 572
SpaceTypeError (class in nipy.core.reference.spaces), 572
spectral_decomposition() (in module nipy.modalities.fmri.fmristat.hrf), 789
sphere_density() (in module nipy.algorithms.clustering.von_mises_fisher_mixture), 219
spherical_search() (in module nipy.algorithms.statistics.rft), 497
split() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 188
split_group() (in module nipy.labs.utils.reproducibility_measures), 720
SPM, 32
SPM approach, 32
SPM software, 32
spm_dispersion_derivative() (in module nipy.modalities.fmri.hemodynamic_models), 811
spm_hrf() (in module nipy.modalities.fmri.hemodynamic_models), 811
spm_path() (nipy.interfaces.spm.SpmInfo static method), 585
spm_time_derivative() (in module nipy.modalities.fmri.hemodynamic_models), 811
SpmInfo (class in nipy.interfaces.spm), 585
square_gaussian_filter() (in module nipy.labs.group.displacement_field), 630
square_gaussian_filter1d() (in module nipy.labs.group.displacement_field), 630
SSE() (nipy.algorithms.statistics.models.nlsmodel.NLSModel method), 458
SSE() (nipy.algorithms.statistics.models.regression.RegressionResults method), 473
SSR() (nipy.algorithms.statistics.models.regression.RegressionResults method), 473
SST() (nipy.algorithms.statistics.models.regression.RegressionResults method), 473
stack2designs() (in module nipy.modalities.fmri.design), 772
stack_contrasts() (in module nipy.modalities.fmri.design), 773
stack_designs() (in module nipy.modalities.fmri.design), 773
step_function() (in module nipy.modalities.fmri.utils), 828
StepFunction (class in nipy.algorithms.statistics.formula.formulae), 804
stratify() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 356
strides_from() (in module nipy.utils.arrays), 849
strip_array_repr() (in module nipy.testing.doctester), 840
StructuredDomain (class in nipy.labs.spatial_models.discrete_domain), 667
subdict() (in module nipy.algorithms.registration.optimizer), 286
subdomain_from_array() (in module nipy.labs.spatial_models.mroi), 691
subdomain_from_balls() (in module nipy.labs.spatial_models.mroi), 691
subdomain_from_image() (in module nipy.labs.spatial_models.mroi), 691
SubDomains (class in nipy.labs.spatial_models.mroi), 685
subfield() (nipy.algorithms.graph.field.Field method), 242
subforest() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 188
subforest() (nipy.algorithms.graph.forest.Forest method), 252
subgraph() (nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method), 189
subgraph() (nipy.algorithms.graph.field.Field method), 242
subgraph() (nipy.algorithms.graph.forest.Forest method), 252
subgraph() (nipy.algorithms.graph.graph.WeightedGraph method), 262
subgrid_affine() (in module nipy.algorithms.registration.affine), 280
subs() (nipy.algorithms.statistics.formula.formulae.Beta method), 353
subs() (nipy.algorithms.statistics.formula.formulae.BetaTerm method), 353
subs() (nipy.algorithms.statistics.formula.formulae.Factor method), 357
subs() (nipy.algorithms.statistics.formula.formulae.FactorTerm method), 383
subs() (nipy.algorithms.statistics.formula.formulae.Formula method), 388
Index 917
<table>
<thead>
<tr>
<th>Function/Method</th>
<th>Module/Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>transpose()</td>
<td>nipy.algorithms.statistics.formula.formulae.FactorTerm method, 385</td>
</tr>
<tr>
<td>transpose()</td>
<td>nipy.algorithms.statistics.formula.formulae.Term method, 417</td>
</tr>
<tr>
<td>transposed_values</td>
<td>nipy.core.reference.array_coders.ArrayCoordMap attribute, 534</td>
</tr>
<tr>
<td>tree_depth()</td>
<td>nipy.algorithms.clustering.hierarchical_clustering.WeightedForest method, 189</td>
</tr>
<tr>
<td>tree_depth()</td>
<td>nipy.algorithms.graph.forest.Forest method, 253</td>
</tr>
<tr>
<td>trigsimp()</td>
<td>nipy.algorithms.statistics.formula.formulae.Beta method, 354</td>
</tr>
<tr>
<td>trigsimp()</td>
<td>nipy.algorithms.statistics.formula.formulae.FactorTerm method, 385</td>
</tr>
<tr>
<td>trigsimp()</td>
<td>nipy.algorithms.statistics.formula.formulae.Term method, 417</td>
</tr>
<tr>
<td>trigsimp()</td>
<td>nipy.algorithms.modalities.fmri.spm.trace.trRV(), 821</td>
</tr>
<tr>
<td>TStat</td>
<td>nipy.algorithms.statistics.rft, 495</td>
</tr>
<tr>
<td>ttest()</td>
<td>nipy.labs.utils.reproducibility_measures, 721</td>
</tr>
<tr>
<td>two_sample_ftest()</td>
<td>nipy.algorithms.statistics.mixed_effects_stat, 426</td>
</tr>
<tr>
<td>two_sample_ttest()</td>
<td>nipy.algorithms.statistics.mixed_effects_stat, 427</td>
</tr>
<tr>
<td>two_sample_test()</td>
<td>nipy.labs.group.permutation_test, 643</td>
</tr>
<tr>
<td>twoSample_test()</td>
<td>nipy.labs.statistical_mapping, 707</td>
</tr>
<tr>
<td>unweighted_likelihood()</td>
<td>nipy.algorithms.clustering.bgmm.BGMM method, 155</td>
</tr>
<tr>
<td>unweighted_likelihood()</td>
<td>nipy.algorithms.clustering.bgmm.VBGMM method, 162</td>
</tr>
<tr>
<td>unweighted_likelihood()</td>
<td>nipy.algorithms.clustering.imm.IMM method, 202</td>
</tr>
<tr>
<td>unweighted_likelihood()</td>
<td>nipy.algorithms.clustering.imm.MixedIMM method, 211</td>
</tr>
<tr>
<td>unweighted_likelihood()</td>
<td>nipy.algorithms.clustering.gmm.GMM method, 177</td>
</tr>
<tr>
<td>unweighted_likelihood()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update()</td>
<td>nipy.algorithms.clustering.bgmm.BGMM method, 155</td>
</tr>
<tr>
<td>update()</td>
<td>nipy.algorithms.clustering.bgmm.VBGMM method, 162</td>
</tr>
<tr>
<td>update()</td>
<td>nipy.algorithms.clustering.imm.IMM method, 202</td>
</tr>
<tr>
<td>update()</td>
<td>nipy.algorithms.clustering.imm.MixedIMM method, 211</td>
</tr>
<tr>
<td>update_block()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update_block_SA()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update_displacements()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update_displacements_SA()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update_labels()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update_mean_effect()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update_mean_effect()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update_means()</td>
<td>nipy.algorithms.clustering.bgmm.BGMM method, 156</td>
</tr>
<tr>
<td>update_means()</td>
<td>nipy.algorithms.clustering.bgmm.VBGMM method, 162</td>
</tr>
<tr>
<td>update_means()</td>
<td>nipy.algorithms.clustering.imm.IMM method, 202</td>
</tr>
<tr>
<td>update_means()</td>
<td>nipy.algorithms.clustering.imm.MixedIMM method, 211</td>
</tr>
<tr>
<td>update_parameters_mcmc()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update_parameters_saem()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
<tr>
<td>update_precisions()</td>
<td>nipy.algorithms.clustering.bgmm.BGMM method, 156</td>
</tr>
<tr>
<td>update_precisions()</td>
<td>nipy.algorithms.clustering.bgmm.VBGMM method, 163</td>
</tr>
<tr>
<td>update_precisions()</td>
<td>nipy.algorithms.clustering.imm.IMM method, 203</td>
</tr>
<tr>
<td>update_summary_statistics()</td>
<td>nipy.labs.group.spatial_relaxation_onesample.multivariate_stat method, 648</td>
</tr>
</tbody>
</table>
z_score() (in module nipy.algorithms.statistics.utils), 500
z_score() (nipy.modalities.fmri.glm.Contrast method), 805
z_suffix (nipy.core.reference.spaces.XYZSpace attribute), 574
z_to_slice() (nipy.algorithms.registration.groupwise_registration.Image4d method), 285
zscore() (in module nipy.labs.utils.zscore), 727
zscore() (nipy.labs.glm.glm.contrast method), 628
zscore() (nipy.labs.group.permutation_test.permutation_test method), 636
zscore() (nipy.labs.group.permutation_test.permutation_test_onesample method), 638
zscore() (nipy.labs.group.permutation_test.permutation_test_onesample_graph method), 640
zscore() (nipy.labs.group.permutation_test.permutation_test_twosample method), 642
zslice() (in module nipy.core.reference.slices), 569
ZSlicer (class in nipy.labs.viz_tools.slicers), 757