# BrainStat

*Release 0.0.1*

**MICA Lab, CNG Lab**

**Nov 03, 2022**

# TABLE OF CONTENTS:

Welcome to BrainStat's documentation!

BrainStat is a toolbox for the statistical analysis and context decoding of neuroimaging data. It implements both univariate and multivariate linear models and interfaces with the Allen Human Brain Atlas and Nimare databases. BrainStat flexibly handles common surface, volume, and parcel level data formats, and provides a series of interactive visualization functions. The toolbox has been implemented in both python and matlab, the two most widely adopted programming languages in the neuroimaging and neuroinformatics communities. It is openly available, and documented here.

CHAPTER

ONE

# DEVELOPERS

- Reinder Vos de Wael - MICA Lab, Montreal Neurological Institute
- Şeyma Bayrak - Max Planck Institute for Human Cognitive and Brain Sciences
- Oualid Benkarim - MICA Lab, Montreal Neurological Institute
- Sara Lariviere - MICA Lab, Montreal Neurological Institute
- Raul Cruces - MICA Lab, Montreal Neurological Institute
- Peer Herholz - Montreal Neurological Institute
- Seok-Jun Hong - Sungkyunkwan University
- Sofie Valk - Max Planck Institute for Human Cognitive and Brain Sciences
- Boris Bernhardt - Montreal Neurological Institute

CHAPTER

TWO

# LICENSE

The BrainStat source code is available under the BSD (3-Clause) license.

CHAPTER

# THREE

# SUPPORT

If you have problems installing the software or questions about usage and documentation, or something else related to BrainStat, you can post to the Issues section of our repository.

## 3.1 Installation Guide

BrainStat is available in Python and MATLAB.

### 3.1.1 Python installation

BrainStat is compatible with Python 3.6-3.8. Compatibility with 3.9 is in the works.

BrainStat can be installed using `pip` (NOTE: This doesn't work yet!):

```
pip install brainstat
```

Alternatively, you can install the package from Github as follows:

```
git clone https://github.com/MICA-LAB/BrainStat.git
cd BrainStat
python setup.py install
```

### 3.1.2 MATLAB installation

This toolbox has been tested with MATLAB versions R2019b and newer.

To install the MATLAB toolbox simply download and unzip the GitHub toolbox and run the following in MATLAB:

```
addpath(genpath('/path/to/BrainSpace/matlab/'))
```

If you want to load BrainStat every time you start MATLAB, type `edit startup` and append the above line to the end of this file.

If you wish to open gifti files you will also need to install the gifti library.

## 3.2 Python Index

### 3.2.1 Tutorials

#### Tutorial 01: Linear Models

In this tutorial you will set up your first linear model with BrainStat. First lets load some example data to play around with. We'll load age, IQ, and left hemispheric cortical thickness for a few subjects.

```
import numpy as np
import nibabel as nib
import os
import brainstat
from brainstat.tutorial.utils import fetch_tutorial_data
from brainstat.context.utils import read_surface_gz
from nilearn.datasets import fetch_surf_fsaverage

brainstat_dir = os.path.dirname(brainstat.__file__)
data_dir = os.path.join(brainstat_dir, "tutorial")

n = 20
tutorial_data = fetch_tutorial_data(n_subjects=n, data_dir=data_dir)
age = tutorial_data["demographics"]["AGE"].to_numpy()
iq = tutorial_data["demographics"]["IQ"].to_numpy()

# Reshape the thickness files such that left and right hemispheres are in the same row.
files = np.reshape(np.array(tutorial_data["image_files"]), (-1, 2))

# We'll use only the left hemisphere in this tutorial.
thickness = np.zeros((n, 10242))
for i in range(n):
    thickness[i, :] = np.squeeze(nib.load(files[i, 0]).get_fdata())
mask = np.all(thickness != 0, axis=0)

pial_left = read_surface_gz(fetch_surf_fsaverage()["pial_left"])
```

```
Dataset created in /home/docs/checkouts/readthedocs.org/user_builds/brainstat/checkouts/
↪latest/brainstat/tutorial/brainstat_tutorial

Downloading data from https://box.bic.mni.mcgill.ca/s/wMPF2vj7EoYWELV/download ...
 ...done. (1 seconds, 0 min)
Downloading data from https://box.bic.mni.mcgill.ca/s/wMPF2vj7EoYWELV/download ...

Downloaded 2015232 of 19209132 bytes (10.5%,    8.7s remaining)
Downloaded 13959168 of 19209132 bytes (72.7%,    0.8s remaining) ...done. (3 seconds, 0␣
↪min)
Extracting data from /home/docs/checkouts/readthedocs.org/user_builds/brainstat/
↪checkouts/latest/brainstat/tutorial/brainstat_tutorial/
↪3c1b224c1c95382411fd038ef5a44916/brainstat_tutorial.zip..... done.
```

Next, we can create a BrainStat linear model by declaring these variables as terms. The term class requires two things: 1) an array or scalar, and 2) a variable name for each column. Lastly, we can create the model by simply adding the terms together.

```
from brainstat.stats.terms import Term

term_intercept = Term(1, names="intercept")
term_age = Term(age, "age")
term_iq = Term(iq, "iq")
model = term_intercept + term_age + term_iq
```

We can also add interaction effects to the model by multiplying terms.

```
model_interaction = term_intercept + term_age + term_iq + term_age * term_iq
```

Now, lets imagine we have some cortical marker (e.g. cortical thickness) for each subject and we want to evaluate whether this marker changes with age whilst correcting for effects of sex and age-sex interactions.

```
from brainstat.stats.SLM import SLM

slm = SLM(model_interaction, -age, surf=pial_left, correction="rft", mask=mask)
slm.fit(thickness)
print(slm.t.shape)  # These are the t-values of the model.
print(slm.P["pval"]["P"])  # These are the random field theory derived p-values.
```

```
(1, 10242)
[1. 1. 1. ... 1. 1. 1.]
```

By default BrainStat uses a two-tailed test. If you want to get a one-tailed test, simply specify it in the SLM model as follows:

```
slm_two_tails = SLM(
    model_interaction, -age, surf=pial_left, correction="rft", two_tailed=False
)
slm_two_tails.fit(thickness)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/brainstat/checkouts/latest/brainstat/
↪stats/_linear_model.py:423: RuntimeWarning: invalid value encountered in divide
  u = Y[i, :, j] / normr
```

Now, imagine that instead of using a fixed effects model, you would prefer a mixed effects model wherein handedness is a random variable. This is simple to set up. All you need to do is initialize the handedness term with the Random class instead, all other code remains identical.

```
from brainstat.stats.terms import Random

random_handedness = Random(tutorial_data["demographics"]["HAND"], name_ran="Handedness")
random_identity = Random(1, name_ran="identity")
model_random = (
    term_intercept
    + term_age
    + term_iq
    + term_age * term_iq
    + random_handedness
    + random_identity
)
```

(continues on next page)

(continued from previous page)

```
slm_random = SLM(model_random, -age, surf=pial_left, correction="fdr", mask=mask)
slm_random.fit(thickness)
```

That concludes the basic usage of the BrainStat for statistical models.

**Total running time of the script:** ( 0 minutes 8.400 seconds)

| | |
|---|---|
| *brainstat* | Neuroimaging statistics toolbox. |

### 3.2.2 brainstat

Neuroimaging statistics toolbox.

#### Modules

| | |
|---|---|
| *brainstat.context* | BrainStat's context decoding module. |
| *brainstat.mesh* | Module for the handling of meshes and mesh data. |
| *brainstat.stats* | The statistics tools of BrainStat |
| *brainstat.tests* | |
| *brainstat.tutorial* | Functions required for the BrainStat Tutorials |

#### brainstat.context

BrainStat's context decoding module.

#### Modules

| | |
|---|---|
| *brainstat.context.genetics* | Genetic decoding using abagen. |
| *brainstat.context.meta_analysis* | Meta-analytic decoding based on NiMARE |
| *brainstat.context.utils* | Utilities for handling label files |

#### brainstat.context.genetics

Genetic decoding using abagen.

#### Functions

| | |
|---|---|
| *surface_genetic_expression*(pial, white, ...) | Computes genetic expression of surface parcels. |

### brainstat.context.genetics.surface_genetic_expression

brainstat.context.genetics.**surface_genetic_expression**(*pial*, *white*, *labels*, *volume_template*, ***, *atlas_info=None*, *ibf_threshold=0.5*, *probe_selection='diff_stability'*, *donor_probes='aggregate'*, *lr_mirror=False*, *exact=True*, *tolerance=2*, *sample_norm='srs'*, *gene_norm='srs'*, *norm_matched=True*, *region_agg='donors'*, *agg_metric='mean'*, *corrected_mni=True*, *reannotated=True*, *return_counts=False*, *return_donors=False*, *donors='all'*, *data_dir=None*, *verbose=1*, *n_proc=1*)

Computes genetic expression of surface parcels.

**Parameters**

- **pial** (*str, BSPolyData, list*) – Path of a pial surface file, BSPolyData of a pial surface or a list containing multiple of the aforementioned.
- **white** (*str, BSPolyData, list*) – Path of a white matter surface file, BSPolyData of a pial surface or a list containing multiple of the aforementioned.
- **labels** (*str, numpy.ndarray, list*) – Path to a label file for the surfaces, numpy array containing the labels, or a list containing multiple of the aforementioned.
- **volume_template** (*str, nibabel.nifti1.Nifti1Image*) – Path to a nifti file to use as a template for the surface to volume procedure, or a loaded NIfTI image.
- **the** (*For details of the remaining parameters please consult*) –
- **"atlas"** (*abagen.get_expression_data() documentation. All its parameters bar*) –
- **parameters.** (*are valid input*) –

**Returns**

*pandas.DataFrame* – Dataframe containing the expression of each gene within each region.

#### Notes

An equal number of pial/white surfaces and labels must be provided. If parcellations overlap across surfaces, then the labels are kept for the first provided surface.

#### Examples

```
>>> from brainstat.context.genetics import surface_genetic_expression
>>> from nilearn import datasets
```

```
>>> fsaverage = datasets.fetch_surf_fsaverage()
>>> destrieux_atlas = datasets.fetch_atlas_surf_destrieux()
>>> parcellation = destrieux_atlas['map_left']
>>> mni152 = datasets.load_mni152_template()
>>> surface_genetic_expression(fsaverage['pial_left'], fsaverage['white_left'],
...    parcellation, mni152)
```

### brainstat.context.meta_analysis

Meta-analytic decoding based on NiMARE

### Functions

| | |
|---|---|
| *fetch_neurosynth_dataset*(data_dir[, ...]) | Downloads the Neurosynth dataset |
| *fetch_nimare_dataset*(data_dir[, keep_neurosynth]) | Downloads the nimare dataset and fetches its path. |
| *surface_decode_nimare*(pial, white, ...[, ...]) | Meta-analytic decoding of surface maps using NeuroSynth or Brainmap. |

### brainstat.context.meta_analysis.fetch_neurosynth_dataset

brainstat.context.meta_analysis.**fetch_neurosynth_dataset**(*data_dir*, *return_pkl=True*, *verbose=False*)

Downloads the Neurosynth dataset

**Parameters**

- **data_dir** (*str*) – Directory in which to download the dataset.
- **return_pkl** (*bool*) – If true, creates and returns the .pkl file. Otherwise returns the dataset and features files.
- **verbose** (*bool*) – If true prints additional output to the console, by default False.

**Returns**

*tuple, str* – If save_pkl is false, returns a tuple containing the path to the database.txt and the features.txt file. Otherwise returns the path to the .pkl file.

### brainstat.context.meta_analysis.fetch_nimare_dataset

brainstat.context.meta_analysis.**fetch_nimare_dataset**(*data_dir*, *keep_neurosynth=False*)

Downloads the nimare dataset and fetches its path.

**Parameters**

- **data_dir** (*str*) – Path to the directory where the dataset will be saved.
- **keep_neurosynth** (*bool, optional*) – If true, then the neurosynth data files are kept, by default False. Note that this will not delete existing neurosynth files.

**Returns**

*nimare.Dataset* – Downloaded NiMARE dataset.

### brainstat.context.meta_analysis.surface_decode_nimare

brainstat.context.meta_analysis.**surface_decode_nimare**(*pial*, *white*, *stat_labels*, *mask_labels*, *interpolation='linear'*, *data_dir=None*, *verbose=True*, *correction='fdr_bh'*, *feature_group=None*, *features=None*)

Meta-analytic decoding of surface maps using NeuroSynth or Brainmap.

**Parameters**

- **pial** (*str, BSPolyData, list*) – Path of a pial surface file, BSPolyData of a pial surface or a list containing multiple of the aforementioned.
- **white** (*str, BSPolyData, list*) – Path of a white matter surface file, BSPolyData of a pial surface or a list containing multiple of the aforementioned.
- **stat_labels** (*str, numpy.ndarray, list*) – Path to a label file for the surfaces, numpy array containing the labels, or a list containing multiple of the aforementioned.
- **mask_labels** (*str, numpy.ndarray, list*) – Path to a mask file for the surfaces, numpy array containing the mask, or a list containing multiple of the aforementioned. If None all vertices are included in the mask. Defaults to None.
- **interpolation** (*str, optional*) – Either 'nearest' for nearest neighbor interpolation, or 'linear' for trilinear interpolation, by default 'linear'.
- **data_dir** (*str, optional*) – The directory of the nimare dataset. If none exists, a new dataset will be downloaded and saved to this path. If None, the directory defaults to your home directory, by default None.
- **verbose** (*bool, optional*) – If true prints additional output to the console, by default True.
- **correction** (*str, optional*) – Multiple comparison correction. Valid options are None and 'fdr_bh', by default 'fdr_bh'.

**Returns**

*pandas.DataFrame* – Table with each label and the following values associated with each label: 'pForward', 'zForward', 'likelihoodForward','pReverse', 'zReverse', and 'probReverse'.

### brainstat.context.utils

Utilities for handling label files

## Functions

| | |
|---|---|
| *combine_parcellations*(files, output_file) | Combines multiple nifti files into one. |
| *load_enigma_histology*(parcellation[, n]) | Loads MPC gradient from the enigma toolbox. |
| *load_mesh_labels*(label_file[, as_int]) | Loads a .label.gii or .csv file. |
| *mutli_surface_to_volume*(pial, white, ...[, ...]) | Interpolates multiple surfaces to the volume. |
| *read_surface_gz*(filename) | Extension of brainspace's read_surface to include .gz files. |

### brainstat.context.utils.combine_parcellations

brainstat.context.utils.**combine_parcellations**(*files*, *output_file*)

Combines multiple nifti files into one.

**Parameters**

- **files** (*list*) – List of strings containing the paths to nifti files.
- **output_file** (*str*) – Path to the output file.

#### Notes

This function assumes that 0's are missing data. When multiple files have non-zero values in the same voxel, then the data from the first provided file is kept.

### brainstat.context.utils.load_enigma_histology

brainstat.context.utils.**load_enigma_histology**(*parcellation*, *n=None*)

Loads MPC gradient from the enigma toolbox.

**Parameters**

- **parcellation** (*str*) – Name of a parcellation. Valid values are: 'aparc', 'glasser', 'schaefer'.
- **n** (*int, optional*) – Number of regions in the parcellation. Only used for schaefer parcellations. Valid values are 100, 200, 300, 400. By default None.

**Returns**

*numpy.array* – BigBrain derived microstructural profile covariance gradient 1.

#### Notes

This function is likely to be removed in a future update. It is strongly discouraged to use this function.

### brainstat.context.utils.load_mesh_labels

brainstat.context.utils.**load_mesh_labels**(*label_file*, *as_int=True*)

Loads a .label.gii or .csv file.

**Parameters**

- **label_file** (*str*) – Path to the label file.
- **as_int** (*bool*) – Determines whether to enforce integer format on the labels, defaults to True.

**Returns**

*numpy.array* – Labels in the file.

### brainstat.context.utils.mutli_surface_to_volume

brainstat.context.utils.**mutli_surface_to_volume**(*pial*, *white*, *volume_template*, *labels*, *output_file*, *interpolation='nearest'*, *verbose=True*)

Interpolates multiple surfaces to the volume.

**Parameters**

- **pial** (*str, BSPolyData, list*) – Path of a pial surface file, BSPolyData of a pial surface or a list containing multiple of the aforementioned.
- **white** (*str, BSPolyData, list*) – Path of a white matter surface file, BSPolyData of a pial surface or a list containing multiple of the aforementioned.
- **labels** (*str, numpy.ndarray, list*) – Path to a label file for the surfaces, numpy array containing the labels, or a list containing multiple of the aforementioned.
- **output_file** (*str*) – Path to the output file, must end in .nii or .nii.gz.
- **volume_template** (*str, nibabel.nifti1.Nifti1Image*) – Path to a nifti file to use as a template for the surface to volume procedure, or a loaded NIfTI image.
- **interpolation** (*str*) – Either 'nearest' for nearest neighbor interpolation, or 'linear' for trilinear interpolation, defaults to 'nearest'.
- **verbose** (*boolean*) – If true, returns verbose output to console, defaults to true.

#### Notes

An equal number of pial/white surfaces and labels must be provided. If parcellations overlap across surfaces, then the labels are kept for the first provided surface.

### brainstat.context.utils.read_surface_gz

brainstat.context.utils.**read_surface_gz**(*filename*)

Extension of brainspace's read_surface to include .gz files.

**Parameters**

**filename** (*str*) – Filename of file to open.

**Returns**

*BSPolyData* – Surface mesh.

### brainstat.mesh

Module for the handling of meshes and mesh data.

### Modules

| | |
|---|---|
| *brainstat.mesh.data* | Operations on data on a mesh. |
| *brainstat.mesh.interpolate* | |
| *brainstat.mesh.utils* | Operations on meshes. |

### brainstat.mesh.data

Operations on data on a mesh.

### Functions

| | |
|---|---|
| *mesh_normalize*(Y[, mask, subdiv]) | Normalizes by subtracting the global mean, or dividing it. |
| *mesh_smooth*(Y, surf, FWHM) | Smooths surface data by repeatedly averaging over edges. |
| *mesh_standardize*(Y[, mask, subdiv]) | Standardizes by subtracting the global mean, or dividing it. |

### brainstat.mesh.data.mesh_normalize

brainstat.mesh.data.**mesh_normalize**(*Y*, *mask=None*, *subdiv='s'*)

Normalizes by subtracting the global mean, or dividing it.

**Parameters**

- **Y** (*numpy array of shape (n x v) or (n x v x k)*) – Data to be normalized.
- **mask** (*numpy boolean array of shape (1 x v), optional*) – True is included, False is excluded. If None, no mask is applied, by default 'None'.
- **subdiv** (*str, optional*) – If 's', demeans Y; if 'd' standardizes to mean 0, standard deviation 100.

**Returns**

- **Y** (*numpy array of shape (n x v) or (n x v x k)*) – Normalized data.
- **Yav** (*numpy array of shape (n x 1) or (n x k)*) – Mean of the input Y along the mask.

### brainstat.mesh.data.mesh_smooth

brainstat.mesh.data.**mesh_smooth**(*Y*, *surf*, *FWHM*)

Smooths surface data by repeatedly averaging over edges.

**Parameters**

- **Y** (*numpy array of shape (n,v) or (n,v,k)*) – surface data, v=#vertices, n=#observations, k=#variates.

- **surf** (*a dictionary with key 'tri' or 'lat', or a BSPolyData object.*) – surf['tri'] = numpy array of shape (t,3), triangle indices, or surf['lat'] = numpy array of shape (nx,ny,nz), 1=in, 0=out, (nx,ny,nz) = size(volume).
- **FWHM** (*approximate FWHM of Gaussian smoothing filter, in mesh units.*) –

**Returns**

**Y** (*numpy array of shape (n,v) or (n,v,k),*) – smoothed data.

### brainstat.mesh.data.mesh_standardize

brainstat.mesh.data.**mesh_standardize**(*Y*, *mask=None*, *subdiv='s'*)

Standardizes by subtracting the global mean, or dividing it.

**Parameters**

- **Y** (*numpy array of shape (n x v)*) – Data to be standardized.
- **mask** (*numpy boolean array of shape (1 x v), optional*) – True is included, False is excluded. If None, no mask is applied, by default 'None'.
- **subdiv** (*str, optional*) – If 's', demeans Y; if 'd' standardizes to mean 0, standard deviation 100.

**Returns**

- **Y** (*numpy array of shape (n x v)*) – Standardized data.
- **Ym** (*numpy array of shape (n x 1)*) – Mean of the input Y along the mask.

### brainstat.mesh.interpolate

### Functions

| | |
|---|---|
| *cortical_ribbon*(pial_mesh, wm_mesh, nii[, ...]) | Finds voxels inside of the cortical ribbon. |
| *ribbon_interpolation*(pial_mesh, wm_mesh, ...) | Performs label interpolation in the cortical ribbon. |
| *surface_to_volume*(pial_mesh, wm_mesh, ...[, ...]) | Projects surface labels to the cortical ribbon. |

### brainstat.mesh.interpolate.cortical_ribbon

brainstat.mesh.interpolate.**cortical_ribbon**(*pial_mesh*, *wm_mesh*, *nii*, *mesh_distance=6*, *verbose=False*)

Finds voxels inside of the cortical ribbon.

**Parameters**

- **pial_mesh** (*BSPolyData*) – Pial mesh.
- **wm_mesh** (*BSPolyData*) – White matter mesh.
- **nii** (*Nibabel nifti*) – Nifti image containing the space in which to output the ribbon.
- **mesh_distance** (*int, optional*) – Maximum distance from the cortical mesh at which the ribbon may occur. Used to reduce the search space, by default 6.
- **verbose** (*bool*) – If True, returns printed output, defaults to False.

**Returns**
*numpy.array* – Matrix coordinates of voxels inside the cortical ribbon.

### brainstat.mesh.interpolate.ribbon_interpolation

brainstat.mesh.interpolate.**ribbon_interpolation**(*pial_mesh*, *wm_mesh*, *labels*, *nii*, *points*, *interpolation='nearest'*)

Performs label interpolation in the cortical ribbon.

**Parameters**

- **pial_mesh** (*BSPolyData*) – Pial mesh.
- **wm_mesh** (*BSPolydata*) – White matter mesh.
- **labels** (*str, numpy.ndarray*) – Filename of a .label.gii or .shape.gii file, or a numpy array containing the labels.
- **nii** (*Nibabel nifti*) – Reference nifti image.
- **points** (*numpy.array*) – Numpy array containing the coordinates of the ribbon.

**Returns**
*numpy.array* – Interpolated value for each input point.

#### Notes

Strictly, this function will work outside the cortical ribbon too and assign any point to its label on the nearest mesh. An adventurous user could use this for nearest neighbour surface to volume anywhere in the brain, although such usage is not offically supported.

### brainstat.mesh.interpolate.surface_to_volume

brainstat.mesh.interpolate.**surface_to_volume**(*pial_mesh*, *wm_mesh*, *labels*, *volume_template*, *volume_save*, *interpolation='nearest'*, *verbose=False*)

Projects surface labels to the cortical ribbon.

**Parameters**

- **pial_mesh** (*str, BSPolyData*) – Filename of a pial mesh or a BSPolyData object of the same.
- **wm_mesh** (*str, BSPolyData*) – Filename of a pial mesh or a BSPolyData object of the same.
- **labels** (*str, numpy.ndarray*) – Filename of a .label.gii or .shape.gii file, or a numpy array containing the labels.
- **volume_template** (*str, nibabel.nifti1.Nifti1Image*) – Filename of a nifti image in the same space as the mesh files or a NIfTI image loaded with nibabel.
- **volume_save** (*str*) – Filename to which the label image will be saved.
- **interpolation** (*str*) – Either 'nearest' for nearest neighbor interpolation, or 'linear' for trilinear interpolation, defaults to 'nearest'.
- **verbose** (*bool*) – If True, returns printed output, defaults to False.

### brainstat.mesh.utils

Operations on meshes.

### Functions

| | |
|---|---|
| *mesh_average*(filenames[, fun, output_surfstat]) | Average, minimum, or maximum of surfaces. |
| *mesh_edges*(surf[, mask]) | Converts the triangles or lattices of a mesh to edges. |

### brainstat.mesh.utils.mesh_average

brainstat.mesh.utils.**mesh_average**(*filenames*, *fun=<ufunc 'add'>*, *output_surfstat=False*)

Average, minimum, or maximum of surfaces.

**Args:**

filenames (2D numpy array): Numpy array of filenames of surfaces or BSPolyData objects.

fun : function handle to apply to two surfaces, e.g. np.add (default) will give the average of the surfaces, np.fmin or np.fmax will give the min or max, respectively.

**output_surfstat (boolean): If True, outputs the surface in SurfStat format. If false**
outputs the surface as BSPolyData. Default is False.

**Returns:**

surface [BSPolyData, dict]: The output surface.

### brainstat.mesh.utils.mesh_edges

brainstat.mesh.utils.**mesh_edges**(*surf*, *mask=None*)

Converts the triangles or lattices of a mesh to edges.

**Args:**

surf (dict): = a dictionary with key 'tri' or 'lat' surf['tri'] = (t x 3) numpy array of triangle indices, t:#triangles, or, surf['lat'] = 3D numpy array of 1's and 0's (1:in, 0:out). or surf (BSPolyData) = a BrainSpace surface object or surf (SLM) = a SLM object with an associated surface.

**Returns:**

edg (np.array): A e-by-2 numpy array containing the indices of the edges, where e is the number of edges.

### brainstat.stats

The statistics tools of BrainStat

### Modules

| | |
|---|---|
| *brainstat.stats.SLM* | Standard Linear regression models. |
| *brainstat.stats.terms* | Classes for fixed, mixed, and random effects. |
| *brainstat.stats.utils* | Utilities for the stats functions. |

### brainstat.stats.SLM

Standard Linear regression models.

### Functions

| | |
|---|---|
| *f_test*(slm1, slm2) | F-statistics for comparing two uni- or multi-variate fixed effects models. |

### brainstat.stats.SLM.f_test

brainstat.stats.SLM.**f_test**(*slm1*, *slm2*)

F-statistics for comparing two uni- or multi-variate fixed effects models.

**Parameters**

- **slm1** (brainstat.stats.SLM.SLM) – Standard linear model returned by the t_test function; see Notes for details.
- **slm2** (brainstat.stats.SLM.SLM) – Standard linear model returned by the t_test function; see Notes for details.

**Returns**

*brainstat.stats.SLM.SLM* – Standard linear model with f-test results included.

### Classes

| | |
|---|---|
| *SLM*(model, contrast[, surf, mask, ...]) | Core Class for running BrainStat linear models |

### brainstat.stats.SLM.SLM

**class** brainstat.stats.SLM.**SLM**(*model*, *contrast*, *surf=None*, *mask=None*, ***, *correction=None*, *niter=1*, *thetalim=0.01*, *drlim=0.1*, *two_tailed=True*, *cluster_threshold=0.001*)

Bases: object

Core Class for running BrainStat linear models

**__init__**(*model*, *contrast*, *surf=None*, *mask=None*, ***, *correction=None*, *niter=1*, *thetalim=0.01*, *drlim=0.1*, *two_tailed=True*, *cluster_threshold=0.001*)

Constructor for the SLM class.

**Parameters**

- **model** (brainstat.stats.terms.Term) – The linear model to be fitted of dimensions (observations, predictors).
- **contrast** (*array-like*, brainstat.stats.terms.Term) – Vector of contrasts in the observations.
- **surf** (*dict*, *BSPolyData*, *optional*) – A surface provided as either a dictionary with keys 'tri' for its faces (n-by-3 array) and 'coord' for its coordinates (3-by-n array), or as a BrainSpace BSPolyData object by default None.
- **mask** (*array-like*, *optional*) – A mask containing True for vertices to include in the analysis, by default None.
- **correction** (*str*, *list*, *optional*) – String or list of strings. If it contains "rft" a random field theory multiple comparisons correction will be run. If it contains "fdr" a false discovery rate multiple comparisons correction will be run. Both may be provided. By default None.
- **niter** (*int*, *optional*) – Number of iterations of the Fisher scoring algorithm for fitting mixed effects models, by default 1.
- **thetalim** (*float*, *optional*) – Lower limit on variance coefficients in standard deviations, by default 0.01.
- **drlim** (*float*, *optional*) – Step of ratio of variance coefficients in standard deviations, by default 0.1.
- **two_tailed** (*bool*, *optional*) – Determines whether to return two-tailed or one-tailed p-values. Note that multivariate analyses can only be two-tailed, by default True.
- **cluster_threshold** (*float*, *optional*) – P-value threshold or statistic threshold for defining clusters in random field theory, by default 0.001.

### Methods

| | |
|---|---|
| *__init__*(model, contrast[, surf, mask, ...]) | Constructor for the SLM class. |
| *fdr*() | Q-values for False Discovey Rate of resels. |
| *fit*(Y) | Fits the SLM model |
| *linear_model*(Y) | Fits linear mixed effects models to surface data and estimates resels. |
| *multiple_comparison_corrections*() | Performs multiple comparisons corrections. |
| *random_field_theory*() | Corrected P-values for vertices and clusters. |
| *t_test*() | T statistics for a contrast in a univariate or multivariate model. |

**fdr**()

Q-values for False Discovey Rate of resels.

**Parameters**
**self** (brainstat.stats.SLM.SLM) – SLM object with computed t-values.

**Returns**
*numpy.array* – Q-values for false discovery rate of resels.

**fit**(*Y*)

Fits the SLM model

**Parameters**
**Y** (*numpy.array*) – Input data (observation, vertex, variate)

**Raises**
**ValueError** – An error will be thrown when multivariate data is provided and a one-tailed test is requested.

`linear_model(Y)`

Fits linear mixed effects models to surface data and estimates resels.

**Parameters**

- **self** (brainstat.stats.SLM.SLM) – Initialized SLM object.
- **Y** (*numpy array,* brainstat.stats.terms.Term) – Input data of shape (samples, vertices, features).

`multiple_comparison_corrections()`

Performs multiple comparisons corrections.

`random_field_theory()`

Corrected P-values for vertices and clusters. :param self: SLM object with computed t-values. :type self: brainstat.stats.SLM.SLM

**Returns**

- **pval** (*a dictionary with keys 'P', 'C', 'mask'.*) –

  **pval['P']**
  [2D numpy array of shape (1,v).] Corrected P-values for vertices.

  **pval['C']**
  [2D numpy array of shape (1,v).] Corrected P-values for clusters.

- **peak** (*a dictionary with keys 't', 'vertid', 'clusid', 'P'.*) –

  **peak['t']**
  [2D numpy array of shape (np,1).] Peaks (local maxima).

  **peak['vertid']**
  [2D numpy array of shape (np,1).] Vertex.

  **peak['clusid']**
  [2D numpy array of shape (np,1).] Cluster id numbers.

  **peak['P']**
  [2D numpy array of shape (np,1).] Corrected P-values for the peak.

- **clus** (*a dictionary with keys 'clusid', 'nverts', 'resels', 'P.'*) –

  **clus['clusid']**
  [2D numpy array of shape (nc,1).] Cluster id numbers

  **clus['nverts']**
  [2D numpy array of shape (nc,1).] Number of vertices in cluster.

  **clus['resels']**
  [2D numpy array of shape (nc,1).] resels in the cluster.

  **clus['P']**
  [2D numpy array of shape (nc,1).] Corrected P-values for the cluster.

- **clusid** (*2D numpy array of shape (1,v).*) – Cluster id's for each vertex.
- **Reference** (*Worsley, K.J., Andermann, M., Koulis, T., MacDonald, D.*)

- *& Evans, A.C. (1999). Detecting changes in nonisotropic images.*
- **Human Brain Mapping, 8** *(98-101.)*

`t_test()`

T statistics for a contrast in a univariate or multivariate model.

**Parameters**

**self** (brainstat.stats.SLM.SLM) – SLM object that has already run linear_model

### brainstat.stats.terms

Classes for fixed, mixed, and random effects.

### Functions

| | |
|---|---|
| *check_duplicate_names*(df1[, df2]) | Check columns with duplicate names. |
| *check_names*(x) | Return True if *x* is Term, Series or DataFrame. |
| *get_index*(df) | Get index for column names of the form x{i}. |
| *remove_duplicate_columns*(df[, tol]) | Remove duplicate columns. |
| *to_df*(x[, n, names, idx]) | Convert input to DataFrame. |

### brainstat.stats.terms.check_duplicate_names

brainstat.stats.terms.**check_duplicate_names**(*df1*, *df2=None*)

Check columns with duplicate names.

**Parameters**

- **df1** (*DataFrame*) – Input dataframe.
- **df2** (*DataFrame, optional*) – If provided, check that dataframes do not contain columns with same names. Default is None.

**Raises**

ValueError – If there are columns with duplicate names.

### brainstat.stats.terms.check_names

brainstat.stats.terms.**check_names**(*x*)

Return True if *x* is Term, Series or DataFrame.

### brainstat.stats.terms.get_index

brainstat.stats.terms.**get_index**(*df*)

Get index for column names of the form x{i}.

If there are none, return 0.

**Parameters**
**df** (*DataFrame*) – Input dataframe.

**Returns**
**index** (*int*) – Index for the next x column. If *df* is empty, return None.

### brainstat.stats.terms.remove_duplicate_columns

brainstat.stats.terms.**remove_duplicate_columns**(*df*, *tol=1e-08*)

Remove duplicate columns.

**Parameters**

- **df** (*DataFrame*) – Input dataframe.
- **tol** (*float, optional*) – Tolerance to assess duplicate columns. Default is 1e-8.

**Returns**
**columns** (*list of str*) – Columns to keep after removing duplicates.

### brainstat.stats.terms.to_df

brainstat.stats.terms.**to_df**(*x*, *n=1*, *names=None*, *idx=None*)

Convert input to DataFrame.

**Parameters**

- **x** (*array-like or Term*) – Input data.
- **n** (*int, optional*) – If input is a scalar, broadcast to column of *n* entries. Default is 1.
- **names** (*str, list of str or None, optional*) – Names for each column in *x*. Default is None.
- **idx** (*int or None, optional*) – Staring index for variable names of the for *x{i}*.

**Returns**
**df** (*DataFrame*) – Input *x* wrapped in a DataFrame.

## Classes

| | |
|---|---|
| *Random*([ran, fix, name_ran, name_fix, ranisvar]) | Build a random term object for a linear model. |
| *Term*([x, names]) | Build a term object for a linear model. |

### brainstat.stats.terms.Random

**class** brainstat.stats.terms.**Random**(*ran=None*, *fix=None*, *name_ran=None*, *name_fix=None*, *ranisvar=False*)

Bases: object

Build a random term object for a linear model.

**Parameters**

- **ran** (*array-like or DataFrame, optional*) – For the random effects. If None, the random term is empty. Default is None.
- **fix** (*array-like or DataFrame, optional*) – If None, the fixed effects.
- **name_ran** (*str, optional*) – Name for the random term. If None, it defauts to 'xi'. Default is None.
- **name_fix** (*str, optional*) – Name for the *fix* term. If None, it defauts to 'xi'. Default is None.
- **ranisvar** (*bool, optional*) – If True, *ran* is already a term for the variance. Default is False.

**Variables**

- **mean** (Term) – Term for the mean.
- **variance** (Term) – Term for the variance.

**See also:**

*Term*
Term object

#### Examples

```
>>> r = Random()
>>> r.is_empty
True
```

```
>>> r2 = Random(np.arange(5), name_ran='r1')
>>> r2.mean.is_empty
True
>>> r2.variance.shape
(25, 1)
```

**__init__**(*ran=None*, *fix=None*, *name_ran=None*, *name_fix=None*, *ranisvar=False*)

### Methods

| |
|---|
| `__init__`([ran, fix, name_ran, name_fix, ...]) |
| `broadcast_to`(r1, r2) |

### Attributes

| |
|---|
| `empty` |
| `shape` |

## brainstat.stats.terms.Term

**class** brainstat.stats.terms.**Term**(*x=None*, *names=None*)

Bases: `object`

Build a term object for a linear model.

**Parameters**

- **x** (*array-like or DataFrame, optional*) – If None, the term is empty. Default is None.
- **names** (*str or list of str, optional*) – Names for each column in *x*. If None, it defauts to {'x0', 'x1', ...}. Default is None.

**Variables**

- **x** (*DataFrame*) – Design matrix.
- **names** (*list of str*) – Names of columns in the design matrix.

**See also:**

*Random*
Random term

### Examples

```
>>> t = Term()
>>> t.is_empty
True
```

```
>>> t1 = Term(np.arange(5), names='t1')
>>> t2 = Term(np.random.randn(5, 1), names=['t2'])
>>> t3 = t1 + t2 + 1
>>> t3.shape
(5, 3)
```

**__init__**(*x=None*, *names=None*)

### Methods

| | |
|---|---|
| *__init__*([x, names]) | |

### Attributes

| | |
|---|---|
| is_empty | |
| is_scalar | |
| matrix | |
| names | |
| tolerance | |

## brainstat.stats.utils

Utilities for the stats functions.

### Functions

| | |
|---|---|
| *apply_mask*(Y, mask[, axis]) | Masks the data along a specified axis |
| *colon*(start, stop[, increment]) | Generates a range of numbers including the stop number. |
| *interp1*(x, y, ix[, kind]) | Interpolation between datapoints. |
| *ismember*(A, B[, rows]) | Tests whether elements of A appear in B. |
| *row_ismember*(a, b) | Tests whether rows of a occur in b. |
| *undo_mask*(Y, mask[, axis, missing_value]) | Restores the original dimensions of masked data. |

## brainstat.stats.utils.apply_mask

brainstat.stats.utils.**apply_mask**(*Y*, *mask*, *axis=0*)

Masks the data along a specified axis

**Parameters**

- **Y** (*array-like*) – Data to be masked.
- **mask** (*array-like*) – Boolean vector containing True for each element to keep.
- **axis** (*int, optional*) – Axis along which to operate, by default 0.

**Returns**

*numpy.array* – Masked data.

### brainstat.stats.utils.colon

brainstat.stats.utils.**colon**(*start*, *stop*, *increment=1*)

Generates a range of numbers including the stop number.

**Parameters**

- **start** (*(int)*) – Starting scalar number of the range.
- **stop** (*(int)*) – Stopping scalar number of the range.
- **(float)** (*increment*) – Increments of the range.

**Returns**

*numpy.array* – The requested numbers.

### brainstat.stats.utils.interp1

brainstat.stats.utils.**interp1**(*x*, *y*, *ix*, *kind='linear'*)

Interpolation between datapoints.

**Parameters**

- **x** (*(numpy.array)*) – x coordinates of training data.
- **y** (*(numpy.array)*) – y coordinates of training data.
- **ix** (*(numpy.array)*) – coordinates of the interpolated points.
- **(numpy.array)** (*kind*) – type of interpolation; see scipy.interpolate.interp1d for options.

**Returns**

*numpy.array* – interpolated y coordinates.

### brainstat.stats.utils.ismember

brainstat.stats.utils.**ismember**(*A*, *B*, *rows=False*)

Tests whether elements of A appear in B.

**Parameters**

- **A** (*(numpy.array)*) – 1D or 2D array
- **B** (*(numpy.array)*) – 1D or 2D array
- **rows** (*(logical)*) – If true test for row correspondence rather than element correpondence.

**Returns**

- *logical* – Boolean of the same size as A denoting which elements (or rows) occur in B.
- *numpy.array* – Indices of matching elements/rows in A.

#### Notes

For row-wise comparisons, row_ismember should be significantly faster.

### brainstat.stats.utils.row_ismember

brainstat.stats.utils.**row_ismember**(*a*, *b*)

Tests whether rows of a occur in b.

**Parameters**

- **a** (*numpy.array*) – a 2D array with the same number of columns as b.
- **b** (*numpy.array*) – a 2D array with the same number of columns as a.

**Returns**

*numpy.array* – Indices of rows in a that occur in b.

### brainstat.stats.utils.undo_mask

brainstat.stats.utils.**undo_mask**(*Y*, *mask*, *axis=0*, *missing_value=nan*)

Restores the original dimensions of masked data.

**Parameters**

- **Y** (*array-like*) – Masked data.
- **mask** (*array-like*) – Boolean vector used to mask the data.
- **axis** (*int, optional*) – Axis along which to operate, by default 0.
- **missing_value** (*scalar, optional*) – Number to insert for missing values, by default np.nan.

**Returns**

*numpy.array* – Unmasked data.

### brainstat.tests

### Modules



## brainstat.tests.conftest

### Functions

*pytest_configure*(config)

### brainstat.tests.conftest.pytest_configure

brainstat.tests.conftest.**pytest_configure**(*config*)

### brainstat.tests.test_SLM

### Functions

| | |
|---|---|
| *create_parameter_grid*(samples, predictors) | Creates a parameter grid for the test function. |
| *test_SLM*() | Tests the SLM model using a grid of parameters |

### brainstat.tests.test_SLM.create_parameter_grid

brainstat.tests.test_SLM.**create_parameter_grid**(*samples*, *predictors*)

Creates a parameter grid for the test function.

**Returns**

*ParameterGrid* – All pairings of parameters to be run through the SLM class.

### brainstat.tests.test_SLM.test_SLM

brainstat.tests.test_SLM.**test_SLM**()

Tests the SLM model using a grid of parameters

**Raises**

Exception – First exception that occurs in computing the SLM.

### brainstat.tests.test_f_test

### Functions

| | |
|---|---|
| *dummy_test*(infile, expfile) | |
| *test_01*() | |
| *test_02*() | |
| *test_03*() | |
| *test_04*() | |
| *test_05*() | |
| *test_06*() | |
| *test_07*() | |
| *test_08*() | |
| *test_09*() | |
| *test_10*() | |
| *test_11*() | |
| *test_12*() | |

### brainstat.tests.test_f_test.dummy_test

brainstat.tests.test_f_test.**dummy_test**(*infile*, *expfile*)

### brainstat.tests.test_f_test.test_01

brainstat.tests.test_f_test.**test_01**()

### brainstat.tests.test_f_test.test_02

brainstat.tests.test_f_test.**test_02**()

#### brainstat.tests.test_f_test.test_03

brainstat.tests.test_f_test.**test_03**()

#### brainstat.tests.test_f_test.test_04

brainstat.tests.test_f_test.**test_04**()

#### brainstat.tests.test_f_test.test_05

brainstat.tests.test_f_test.**test_05**()

#### brainstat.tests.test_f_test.test_06

brainstat.tests.test_f_test.**test_06**()

#### brainstat.tests.test_f_test.test_07

brainstat.tests.test_f_test.**test_07**()

#### brainstat.tests.test_f_test.test_08

brainstat.tests.test_f_test.**test_08**()

#### brainstat.tests.test_f_test.test_09

brainstat.tests.test_f_test.**test_09**()

#### brainstat.tests.test_f_test.test_10

brainstat.tests.test_f_test.**test_10**()

#### brainstat.tests.test_f_test.test_11

brainstat.tests.test_f_test.**test_11**()

### brainstat.tests.test_f_test.test_12

brainstat.tests.test_f_test.**test_12**()

### brainstat.tests.test_fdr

### Functions

| | |
|---|---|
| *dummy_test*(infile, expfile) | |
| *test_01*() | |
| *test_02*() | |
| *test_03*() | |
| *test_04*() | |
| *test_05*() | |
| *test_06*() | |
| *test_07*() | |
| *test_08*() | |
| *test_09*() | |
| *test_10*() | |
| *test_11*() | |
| *test_12*() | |
| *test_13*() | |
| *test_14*() | |

### brainstat.tests.test_fdr.dummy_test

brainstat.tests.test_fdr.**dummy_test**(*infile*, *expfile*)

### brainstat.tests.test_fdr.test_01

brainstat.tests.test_fdr.**test_01**()

### brainstat.tests.test_fdr.test_02

brainstat.tests.test_fdr.**test_02**()

### brainstat.tests.test_fdr.test_03

brainstat.tests.test_fdr.**test_03**()

### brainstat.tests.test_fdr.test_04

brainstat.tests.test_fdr.**test_04**()

### brainstat.tests.test_fdr.test_05

brainstat.tests.test_fdr.**test_05**()

### brainstat.tests.test_fdr.test_06

brainstat.tests.test_fdr.**test_06**()

### brainstat.tests.test_fdr.test_07

brainstat.tests.test_fdr.**test_07**()

### brainstat.tests.test_fdr.test_08

brainstat.tests.test_fdr.**test_08**()

### brainstat.tests.test_fdr.test_09

brainstat.tests.test_fdr.**test_09**()

### brainstat.tests.test_fdr.test_10

brainstat.tests.test_fdr.**test_10**()

### brainstat.tests.test_fdr.test_11

brainstat.tests.test_fdr.**test_11**()

### brainstat.tests.test_fdr.test_12

brainstat.tests.test_fdr.**test_12**()

### brainstat.tests.test_fdr.test_13

brainstat.tests.test_fdr.**test_13**()

### brainstat.tests.test_fdr.test_14

brainstat.tests.test_fdr.**test_14**()

### brainstat.tests.test_linear_model

### Functions

| | |
|---|---|
| *dummy_test*(idic, oslm) | |
| *test_01*() | |
| *test_02*() | |
| *test_03*() | |
| *test_04*() | |
| *test_05*() | |
| *test_06*() | |
| *test_07*() | |
| *test_08*() | |
| *test_09*() | |
| *test_10*() | |
| *test_11*() | |
| *test_12*() | |
| *test_13*() | |
| *test_14*() | |
| *test_15*() | |
| *test_16*() | |

### brainstat.tests.test_linear_model.dummy_test

brainstat.tests.test_linear_model.**dummy_test**(*idic*, *oslm*)

### brainstat.tests.test_linear_model.test_01

brainstat.tests.test_linear_model.**test_01**()

### brainstat.tests.test_linear_model.test_02

brainstat.tests.test_linear_model.**test_02**()

### brainstat.tests.test_linear_model.test_03

brainstat.tests.test_linear_model.**test_03**()

### brainstat.tests.test_linear_model.test_04

brainstat.tests.test_linear_model.**test_04**()

### brainstat.tests.test_linear_model.test_05

brainstat.tests.test_linear_model.**test_05**()

### brainstat.tests.test_linear_model.test_06

brainstat.tests.test_linear_model.**test_06**()

### brainstat.tests.test_linear_model.test_07

brainstat.tests.test_linear_model.**test_07**()

### brainstat.tests.test_linear_model.test_08

brainstat.tests.test_linear_model.**test_08**()

### brainstat.tests.test_linear_model.test_09

brainstat.tests.test_linear_model.**test_09**()

### brainstat.tests.test_linear_model.test_10

brainstat.tests.test_linear_model.**test_10**()

### brainstat.tests.test_linear_model.test_11

brainstat.tests.test_linear_model.**test_11**()

### brainstat.tests.test_linear_model.test_12

brainstat.tests.test_linear_model.**test_12**()

### brainstat.tests.test_linear_model.test_13

brainstat.tests.test_linear_model.**test_13**()

### brainstat.tests.test_linear_model.test_14

brainstat.tests.test_linear_model.**test_14**()

### brainstat.tests.test_linear_model.test_15

brainstat.tests.test_linear_model.**test_15**()

### brainstat.tests.test_linear_model.test_16

brainstat.tests.test_linear_model.**test_16**()

### brainstat.tests.test_mesh_edges

#### Functions

| |
|---|
| *dummy_test*(infile, expfile) |
| *test_01*() |
| *test_02*() |
| *test_03*() |
| *test_04*() |
| *test_06*() |

### brainstat.tests.test_mesh_edges.dummy_test

brainstat.tests.test_mesh_edges.**dummy_test**(*infile*, *expfile*)

### brainstat.tests.test_mesh_edges.test_01

brainstat.tests.test_mesh_edges.**test_01**()

### brainstat.tests.test_mesh_edges.test_02

brainstat.tests.test_mesh_edges.**test_02**()

### brainstat.tests.test_mesh_edges.test_03

brainstat.tests.test_mesh_edges.**test_03**()

### brainstat.tests.test_mesh_edges.test_04

brainstat.tests.test_mesh_edges.**test_04**()

### brainstat.tests.test_mesh_edges.test_06

brainstat.tests.test_mesh_edges.**test_06**()

### brainstat.tests.test_mesh_normalize

#### Functions

| |
|---|
| *dummy_test*(infile, expfile) |
| *test_01*() |
| *test_02*() |
| *test_03*() |
| *test_04*() |

### brainstat.tests.test_mesh_normalize.dummy_test

brainstat.tests.test_mesh_normalize.**dummy_test**(*infile*, *expfile*)

### brainstat.tests.test_mesh_normalize.test_01

brainstat.tests.test_mesh_normalize.**test_01**()

### brainstat.tests.test_mesh_normalize.test_02

brainstat.tests.test_mesh_normalize.**test_02**()

### brainstat.tests.test_mesh_normalize.test_03

brainstat.tests.test_mesh_normalize.**test_03**()

### brainstat.tests.test_mesh_normalize.test_04

brainstat.tests.test_mesh_normalize.**test_04**()

### brainstat.tests.test_mesh_smooth

#### Functions

| |
|---|
| *dummy_test*(infile, expfile) |
| *test_01*() |
| *test_02*() |
| *test_03*() |
| *test_04*() |
| *test_05*() |
| *test_06*() |
| *test_07*() |
| *test_08*() |
| *test_09*() |
| *test_10*() |

### brainstat.tests.test_mesh_smooth.dummy_test

brainstat.tests.test_mesh_smooth.**dummy_test**(*infile*, *expfile*)

### brainstat.tests.test_mesh_smooth.test_01

brainstat.tests.test_mesh_smooth.**test_01**()

### brainstat.tests.test_mesh_smooth.test_02

brainstat.tests.test_mesh_smooth.**test_02**()

### brainstat.tests.test_mesh_smooth.test_03

brainstat.tests.test_mesh_smooth.**test_03**()

### brainstat.tests.test_mesh_smooth.test_04

brainstat.tests.test_mesh_smooth.**test_04**()

### brainstat.tests.test_mesh_smooth.test_05

brainstat.tests.test_mesh_smooth.**test_05**()

### brainstat.tests.test_mesh_smooth.test_06

brainstat.tests.test_mesh_smooth.**test_06**()

### brainstat.tests.test_mesh_smooth.test_07

brainstat.tests.test_mesh_smooth.**test_07**()

### brainstat.tests.test_mesh_smooth.test_08

brainstat.tests.test_mesh_smooth.**test_08**()

### brainstat.tests.test_mesh_smooth.test_09

brainstat.tests.test_mesh_smooth.**test_09**()

### brainstat.tests.test_mesh_smooth.test_10

brainstat.tests.test_mesh_smooth.**test_10**()

### brainstat.tests.test_mesh_standardize

## Functions

| | |
|---|---|
| *dummy_test*(infile, expfile) | |
| *test_01*() | |
| *test_02*() | |
| *test_03*() | |
| *test_04*() | |

### brainstat.tests.test_mesh_standardize.dummy_test

brainstat.tests.test_mesh_standardize.**dummy_test**(*infile*, *expfile*)

### brainstat.tests.test_mesh_standardize.test_01

brainstat.tests.test_mesh_standardize.**test_01**()

### brainstat.tests.test_mesh_standardize.test_02

brainstat.tests.test_mesh_standardize.**test_02**()

### brainstat.tests.test_mesh_standardize.test_03

brainstat.tests.test_mesh_standardize.**test_03**()

### brainstat.tests.test_mesh_standardize.test_04

brainstat.tests.test_mesh_standardize.**test_04**()

### brainstat.tests.test_peak_clus

### Functions

| | |
|---|---|
| *dummy_test*(infile, expfile) | |
| *test_01*() | |
| *test_02*() | |
| *test_03*() | |
| *test_04*() | |
| *test_05*() | |
| *test_06*() | |
| *test_07*() | |
| *test_08*() | |
| *test_09*() | |
| *test_10*() | |
| *test_11*() | |
| *test_12*() | |
| *test_13*() | |
| *test_14*() | |
| *test_15*() | |
| *test_16*() | |
| *test_17*() | |
| *test_18*() | |
| *test_19*() | |
| *test_20*() | |

### brainstat.tests.test_peak_clus.dummy_test

brainstat.tests.test_peak_clus.**dummy_test**(*infile*, *expfile*)

### brainstat.tests.test_peak_clus.test_01

brainstat.tests.test_peak_clus.**test_01**()

### brainstat.tests.test_peak_clus.test_02

brainstat.tests.test_peak_clus.**test_02**()

### brainstat.tests.test_peak_clus.test_03

brainstat.tests.test_peak_clus.**test_03**()

### brainstat.tests.test_peak_clus.test_04

brainstat.tests.test_peak_clus.**test_04**()

### brainstat.tests.test_peak_clus.test_05

brainstat.tests.test_peak_clus.**test_05**()

### brainstat.tests.test_peak_clus.test_06

brainstat.tests.test_peak_clus.**test_06**()

### brainstat.tests.test_peak_clus.test_07

brainstat.tests.test_peak_clus.**test_07**()

### brainstat.tests.test_peak_clus.test_08

brainstat.tests.test_peak_clus.**test_08**()

### brainstat.tests.test_peak_clus.test_09

brainstat.tests.test_peak_clus.**test_09**()

### brainstat.tests.test_peak_clus.test_10

brainstat.tests.test_peak_clus.**test_10**()

### brainstat.tests.test_peak_clus.test_11

brainstat.tests.test_peak_clus.**test_11**()

### brainstat.tests.test_peak_clus.test_12

brainstat.tests.test_peak_clus.**test_12**()

### brainstat.tests.test_peak_clus.test_13

brainstat.tests.test_peak_clus.**test_13**()

### brainstat.tests.test_peak_clus.test_14

brainstat.tests.test_peak_clus.**test_14**()

### brainstat.tests.test_peak_clus.test_15

brainstat.tests.test_peak_clus.**test_15**()

### brainstat.tests.test_peak_clus.test_16

brainstat.tests.test_peak_clus.**test_16**()

### brainstat.tests.test_peak_clus.test_17

brainstat.tests.test_peak_clus.**test_17**()

### brainstat.tests.test_peak_clus.test_18

brainstat.tests.test_peak_clus.**test_18**()

### brainstat.tests.test_peak_clus.test_19

brainstat.tests.test_peak_clus.**test_19**()

### brainstat.tests.test_peak_clus.test_20

brainstat.tests.test_peak_clus.**test_20**()

### brainstat.tests.test_random_field_theory

### Functions

| | |
|---|---|
| *dummy_test*(infile, expfile) | |
| *test_01*() | |
| *test_02*() | |
| *test_03*() | |
| *test_04*() | |
| *test_05*() | |
| *test_06*() | |
| *test_07*() | |
| *test_08*() | |
| *test_09*() | |
| *test_10*() | |
| *test_11*() | |
| *test_13*() | |
| *test_14*() | |
| *test_15*() | |
| *test_16*() | |
| *test_17*() | |
| *test_19*() | |

### brainstat.tests.test_random_field_theory.dummy_test

brainstat.tests.test_random_field_theory.**dummy_test**(*infile*, *expfile*)

### brainstat.tests.test_random_field_theory.test_01

brainstat.tests.test_random_field_theory.**test_01**()

### brainstat.tests.test_random_field_theory.test_02

brainstat.tests.test_random_field_theory.**test_02**()

### brainstat.tests.test_random_field_theory.test_03

brainstat.tests.test_random_field_theory.**test_03**()

### brainstat.tests.test_random_field_theory.test_04

brainstat.tests.test_random_field_theory.**test_04**()

### brainstat.tests.test_random_field_theory.test_05

brainstat.tests.test_random_field_theory.**test_05**()

### brainstat.tests.test_random_field_theory.test_06

brainstat.tests.test_random_field_theory.**test_06**()

### brainstat.tests.test_random_field_theory.test_07

brainstat.tests.test_random_field_theory.**test_07**()

### brainstat.tests.test_random_field_theory.test_08

brainstat.tests.test_random_field_theory.**test_08**()

### brainstat.tests.test_random_field_theory.test_09

brainstat.tests.test_random_field_theory.**test_09**()

### brainstat.tests.test_random_field_theory.test_10

brainstat.tests.test_random_field_theory.**test_10**()

### brainstat.tests.test_random_field_theory.test_11

brainstat.tests.test_random_field_theory.**test_11**()

### brainstat.tests.test_random_field_theory.test_13

brainstat.tests.test_random_field_theory.**test_13**()

### brainstat.tests.test_random_field_theory.test_14

brainstat.tests.test_random_field_theory.**test_14**()

### brainstat.tests.test_random_field_theory.test_15

brainstat.tests.test_random_field_theory.**test_15**()

### brainstat.tests.test_random_field_theory.test_16

brainstat.tests.test_random_field_theory.**test_16**()

### brainstat.tests.test_random_field_theory.test_17

brainstat.tests.test_random_field_theory.**test_17**()

### brainstat.tests.test_random_field_theory.test_19

brainstat.tests.test_random_field_theory.**test_19**()

### brainstat.tests.test_resels

### Functions

| | |
|---|---|
| *dummy_test*(infile, expfile) | |
| *test_01*() | |
| *test_02*() | |
| *test_03*() | |
| *test_04*() | |
| *test_05*() | |
| *test_06*() | |
| *test_07*() | |
| *test_08*() | |
| *test_09*() | |
| *test_10*() | |
| *test_11*() | |
| *test_12*() | |
| *test_13*() | |

#### brainstat.tests.test_resels.dummy_test

brainstat.tests.test_resels.**dummy_test**(*infile*, *expfile*)

#### brainstat.tests.test_resels.test_01

brainstat.tests.test_resels.**test_01**()

#### brainstat.tests.test_resels.test_02

brainstat.tests.test_resels.**test_02**()

### brainstat.tests.test_resels.test_03

brainstat.tests.test_resels.**test_03**()

### brainstat.tests.test_resels.test_04

brainstat.tests.test_resels.**test_04**()

### brainstat.tests.test_resels.test_05

brainstat.tests.test_resels.**test_05**()

### brainstat.tests.test_resels.test_06

brainstat.tests.test_resels.**test_06**()

### brainstat.tests.test_resels.test_07

brainstat.tests.test_resels.**test_07**()

### brainstat.tests.test_resels.test_08

brainstat.tests.test_resels.**test_08**()

### brainstat.tests.test_resels.test_09

brainstat.tests.test_resels.**test_09**()

### brainstat.tests.test_resels.test_10

brainstat.tests.test_resels.**test_10**()

### brainstat.tests.test_resels.test_11

brainstat.tests.test_resels.**test_11**()

### brainstat.tests.test_resels.test_12

brainstat.tests.test_resels.**test_12**()

### brainstat.tests.test_resels.test_13

brainstat.tests.test_resels.**test_13**()

### brainstat.tests.test_stat_threshold

#### Functions

| |
|---|
| *dummy_test*(infile, expfile) |
| *test_01*() |
| *test_02*() |
| *test_03*() |
| *test_04*() |
| *test_05*() |
| *test_06*() |
| *test_07*() |
| *test_08*() |
| *test_09*() |
| *test_10*() |
| *test_11*() |
| *test_12*() |
| *test_13*() |
| *test_14*() |
| *test_15*() |
| *test_16*() |

### brainstat.tests.test_stat_threshold.dummy_test

brainstat.tests.test_stat_threshold.**dummy_test**(*infile*, *expfile*)

### brainstat.tests.test_stat_threshold.test_01

brainstat.tests.test_stat_threshold.**test_01**()

### brainstat.tests.test_stat_threshold.test_02

brainstat.tests.test_stat_threshold.**test_02**()

### brainstat.tests.test_stat_threshold.test_03

brainstat.tests.test_stat_threshold.**test_03**()

### brainstat.tests.test_stat_threshold.test_04

brainstat.tests.test_stat_threshold.**test_04**()

### brainstat.tests.test_stat_threshold.test_05

brainstat.tests.test_stat_threshold.**test_05**()

### brainstat.tests.test_stat_threshold.test_06

brainstat.tests.test_stat_threshold.**test_06**()

### brainstat.tests.test_stat_threshold.test_07

brainstat.tests.test_stat_threshold.**test_07**()

### brainstat.tests.test_stat_threshold.test_08

brainstat.tests.test_stat_threshold.**test_08**()

### brainstat.tests.test_stat_threshold.test_09

brainstat.tests.test_stat_threshold.**test_09**()

### brainstat.tests.test_stat_threshold.test_10

brainstat.tests.test_stat_threshold.**test_10**()

### brainstat.tests.test_stat_threshold.test_11

brainstat.tests.test_stat_threshold.**test_11**()

### brainstat.tests.test_stat_threshold.test_12

brainstat.tests.test_stat_threshold.**test_12**()

### brainstat.tests.test_stat_threshold.test_13

brainstat.tests.test_stat_threshold.**test_13**()

### brainstat.tests.test_stat_threshold.test_14

brainstat.tests.test_stat_threshold.**test_14**()

### brainstat.tests.test_stat_threshold.test_15

brainstat.tests.test_stat_threshold.**test_15**()

### brainstat.tests.test_stat_threshold.test_16

brainstat.tests.test_stat_threshold.**test_16**()

### brainstat.tests.test_t_test

### Functions

| | |
|---|---|
| *dummy_test*(infile, expfile) | |
| *test_01*() | |
| *test_02*() | |
| *test_03*() | |
| *test_04*() | |
| *test_05*() | |
| *test_06*() | |
| *test_07*() | |
| *test_08*() | |
| *test_09*() | |

#### brainstat.tests.test_t_test.dummy_test

brainstat.tests.test_t_test.**dummy_test**(*infile*, *expfile*)

#### brainstat.tests.test_t_test.test_01

brainstat.tests.test_t_test.**test_01**()

#### brainstat.tests.test_t_test.test_02

brainstat.tests.test_t_test.**test_02**()

#### brainstat.tests.test_t_test.test_03

brainstat.tests.test_t_test.**test_03**()

### brainstat.tests.test_t_test.test_04

brainstat.tests.test_t_test.**test_04**()

### brainstat.tests.test_t_test.test_05

brainstat.tests.test_t_test.**test_05**()

### brainstat.tests.test_t_test.test_06

brainstat.tests.test_t_test.**test_06**()

### brainstat.tests.test_t_test.test_07

brainstat.tests.test_t_test.**test_07**()

### brainstat.tests.test_t_test.test_08

brainstat.tests.test_t_test.**test_08**()

### brainstat.tests.test_t_test.test_09

brainstat.tests.test_t_test.**test_09**()

### brainstat.tests.testutil

#### Functions

*datadir*(file)

### brainstat.tests.testutil.datadir

brainstat.tests.testutil.**datadir**(*file*)

### brainstat.tutorial

Functions required for the BrainStat Tutorials

### Modules

| |
|---|
| *brainstat.tutorial.utils* |

### brainstat.tutorial.utils

### Functions

| | |
|---|---|
| *fetch_tutorial_data*([n_subjects, data_dir, ...]) | Download and load the surfstat tutorial dataset. |

### brainstat.tutorial.utils.fetch_tutorial_data

brainstat.tutorial.utils.**fetch_tutorial_data**(*n_subjects=20*, *data_dir=None*, *resume=True*, *verbose=1*)

Download and load the surfstat tutorial dataset.

**Parameters**

- **n_subjects** (*int, optional*) – The number of subjects to load from maximum of 100 subjects. By default, 20 subjects will be loaded. If None is given, all 100 subjects will be loaded.
- **data_dir** (*string, optional*) – Path of the data directory. Used to force data storage in a specified location. If None, data will be download to ~ (home directory). Default: None
- **resume** (*bool, optional*) – If true, try resuming download if possible

**Returns**

**data** (*sklearn.datasets.base.Bunch*) –

**Dictionary-like object, the interest attributes are :**

- 'image_files': Paths to image files in mgh format
- 'demographics': Path to CSV file containing demographic information

### References

**Download**
https://box.bic.mni.mcgill.ca/s/wMPF2vj7EoYWELV

## 3.3 MATLAB Index

### 3.3.1 volume_viewer

#### Synopsis

Plots data in a volume (source code).

#### Usage

```
obj = volume_viewer(structural,overlay,varargin);
```

- *structural*: volume of a structural image.
- *overlay*: (Optional argument) an overlay to plot over the image volume, must have identical dimensions.
- *varargin*: Name-Value Pairs (see below).
- *obj*: an object used for the manipulation of the volume viewer.

#### Description

BrainStat's volume viewer allows for the viewing of statistcal maps in volume space. The volume viewer allows for scrolling and clicking through the volume. Note that slice orientation is currently hardcoded and cannot be modified.

#### Name-Value Pairs

- *remove_zero*: Does not display zeros in the overlay (default: true).
- *threshold_lower*: Does not display values of the overlay below the minimum of the colorbar (default: false).
- *threshold_upper*: Does not display values of the overlay above the maximum of the colorbar (default: false).

#### Object Properties

- *slices*: Contains the curently displayed slices.
- *threshold_lower*: See name-value pair with identical name.
- *threshold_upper*: See name-value pair with identical name.
- *remove_zero*: See name-value pair with identical name.
- *image*: Contains the input image (not modifiable).
- *overlay*: Contains the input overlay (not modifiable).
- *handles* Contains handles to the graphics objects (not modifiable).

#### Object Methods

- obj.colorlimits(limits,image): Sets the color limits of the structural (image="image") or overlay (image="overlay") to limits.
- obj.colormap(cmap,image): Sets the color map of the structural (image="image") or overlay (image="overlay") to cmap.

## 3.4 Funding

Our research is kindly supported by:

- Canadian Institutes of Health Research (CIHR)
- National Science and Engineering Research Council of Canada (NSERC)
- Azrieli Center for Autism Research
- The Montreal Neurological Institute]
- Canada Research Chairs Program
- BrainCanada
- SickKids Foundation
- Helmholtz Foundation

We also like to thank these funders for training/salary support

- Savoy Foundation for Epilepsy (to RvdW)
- Healthy Brain and Healthy Lives (to OB)
- Fonds de la Recherche du Quebec - Sante (to BB)

## 3.5 Credits

Some references that are incorporated into BrainStat

### 3.5.1 SurfStat references

Worsley KJ et al. (2009) A Matlab toolbox for the statistical analysis of univariate and multivariate surface and volumetric data using linear mixed effects models and random field theory. NeuroImage, Volume 47, Supplement 1, July 2009, Pages S39-S41. https://doi.org/10.1016/S1053-8119(09)70882-1

Chung MK et al. (2010) General Multivariate Linear Modeling of Surface Shapes Using SurfStat Neuroimage. 53(2):491-505. doi: 10.1016/j.neuroimage.2010.06.032

### 3.5.2 Random field theory references

Adler RJ and Taylor JE (2007). Random fields and geometry. Springer. Hagler DJ Saygin AP and Sereno MI (2006). Smoothing and cluster thresholding for cortical surface-based group analysis of fMRI data. NeuroImage, 33:1093-1103.

Hayasaka S, Phan KL, Liberzon I, Worsley KJ and Nichols TE (2004). Non-Stationary cluster size inference with random field and permutation methods. NeuroImage, 22:676-687.

Taylor JE and Adler RJ (2003), Euler characteristics for Gaussian fields on manifolds. Annals of Probability, 31:533-563.

Taylor JE and Worsley KJ (2007). Detecting sparse signal in random fields, with an application to brain mapping. Journal of the American Statistical Association, 102:913-928.

Worsley KJ, Andermann M, Koulis T, MacDonald D, and Evans AC (1999). Detecting changes in non-isotropic images. Human Brain Mapping, 8:98-101.

### 3.5.3 Multivariate associative techniques

### 3.5.4 Contextualization

# PYTHON MODULE INDEX

## b

# INDEX

## Symbols



## A



## B



## C

## D



## F



## G



## I



## L



## M



## P



## R

## S



## T

## U