BrainStat

Release 0.3.4

MICA Lab, CNG Lab

Feb 18, 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 BigBrain Atlas, 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 requires Python 3.7+. Assuming you have the correct version of Python installed and aliased to *python*, you can install BrainStat by running the following

```
python -m pip install brainstat
```

Python Dependencies

If you want to use the meta analysis module, you'll also have to download and install the package pyembree. This package is only available through conda-forge:

```
conda install -c conda-forge pyembree
```

3.1.2 MATLAB installation

This toolbox is compatible with MATLAB versions R2019b and newer.

We recommend installing the toolbox through the Mathworks FileExchange. Simply download the file as a toolbox and open the .mltbx file in MATLAB. Alternatively, you can install the same .mltbx file from our GitHub Releases.

If you don't want to install BrainStat as a MATLAB Toolbox, you can also simply download the repository and run the following in MATLAB:

```
addpath(genpath('/path/to/BrainStat/brainstat_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.

MATLAB Dependencies

BrainStat relies on functionality included in the BrainSpace toolbox. Please see the BrainSpace installation guide for installation instructions.

If you wish to open gifti files (required for data loader function) you will also need to install the gifti library.

3.2 Python Index

3.2.1 Tutorials

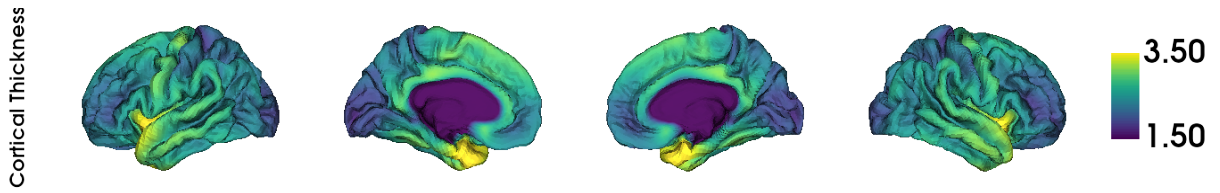
Tutorial 01: Linear Models

In this tutorial you will set up your first linear model with BrainStat. To this end, we will first load some sample data from the MICS dataset.

```
from brainstat.datasets import fetch_mask, fetch_template_surface
from brainstat.tutorial.utils import fetch_mics_data

# Load behavioral markers
thickness, demographics = fetch_mics_data()
pial_left, pial_right = fetch_template_surface("fsaverage5", join=False)
pial_combined = fetch_template_surface("fsaverage5", join=True)
mask = fetch_mask("fsaverage5")
```

Lets have a look at the cortical thickness data. To do this, we will use the surface plotter included with BrainSpace. As we'll be plotting data onto these hemispheres quite often in this tutorial we'll create a simple function for it and plot mean thickness here.

```
import numpy as np
from brainspace.plotting import plot_hemispheres


def local_plot_hemispheres(values, label_text, color_range, cmap="viridis"):
    # Plot cortical surfaces with values as the data, label_text as
    # the labels, and color_range as the limits of the color bar.
    return plot_hemispheres(
        pial_left,
        pial_right,
        values,
        color_bar=True,
        color_range=color_range,
        label_text=label_text,
        cmap=cmap,
        embed_nb=True,
        size=(1400, 200),
        zoom=1.45,
        nan_color=(0.7, 0.7, 0.7, 1),
        cb__labelTextProperty={"fontSize": 12},
        interactive=False,
    )


local_plot_hemispheres(np.mean(thickness, axis=0), ["Cortical Thickness"], (1.5, 3.5))
```



Out:

```
<IPython.core.display.Image object>
```

Lets also have a look at what's inside the demographics data.

```
print(demographics)
```

Out:

```
    SUB_ID  VISIT  AGE_AT_SCAN SEX
0   031404      1    -0.579678   F
1   04a144      1    -0.812116   M
2   0b78f1      1     0.117636   M
3   0d26b9      1     0.466293   F
4   1988b8      1    -0.114802   M
..     ...    ...          ...  ..
77  f25714      1    -0.231021   F
78  f25714      2     0.117636   F
79  f615a5      1    -0.695897   F
80  feac6b      1    -0.695897   F
81  feac6b      2    -0.347240   F

[82 rows x 4 columns]
```

Demographics contains four variables: a subject ID, a visit number (some subjects visited multiple times), their age at the time of scanning and their sex. Lets also print some summary statistics.

```
# Print demographics summary.
for i in range(1, 3):
    print(
        (
            f"Visit {i}, N={np.sum(demographics.VISIT==i)}, "
            f"{np.sum(demographics.SEX[demographics.VISIT == i] == 'F')} females, "
            f"mean subject age {np.mean(demographics.AGE_AT_SCAN[demographics.VISIT
↪== i]):.2f}, "
            f"standard deviation of age: {np.std(demographics.AGE_AT_
↪SCAN[demographics.VISIT==i]):.2f}."
        )
    )
```

Out:

```
Visit 1, N=70, 30 females, mean subject age -0.02, standard deviation of age: 1.02.
Visit 2, N=12, 5 females, mean subject age 0.09, standard deviation of age: 0.84.
```

Next, we will assess whether a subject's age is related to their cortical thickness. To this end we can create a linear model with BrainStat. For our first model, we will only consider the effect of age, i.e. we will disregard the effect of sex and that some subjects visit twice. this end we can create a linear model with BrainStat. First we declare the age variable as a FixedEffect. The FixedEffect class can be created in two ways: either we provide the data with pandas,

as we do here, or we provide a numpy array and a name for the fixed effect. Lets set up the model Y = intercept + B1 * age. Note that BrainStat includes an intercept by default.

```
from brainstat.stats.terms import FixedEffect

term_age = FixedEffect(demographics.AGE_AT_SCAN)
model = term_age
```

As said before, if your data is not in a pandas DataFrame (e.g. numpy), you'll have to provide the name of the effect as an additional parameter as follows:

```
term_age_2 = FixedEffect(demographics.AGE_AT_SCAN.to_numpy(), "AGE_AT_SCAN")
```

Lets have a look at one of these models. As you can see below, the model is stored in a format closely resembling a pandas DataFrame. Note that an intercept is automatically added to the model. This behavior can be disabled in the FixedEffect call, but we recommend leaving it enabled. We can also access the vectors related to each effect by their name i.e. model.intercept and model.AGE_AT_SCAN will return the vectors of the intercept and age, respectively.

```
print(model)
```

Out:

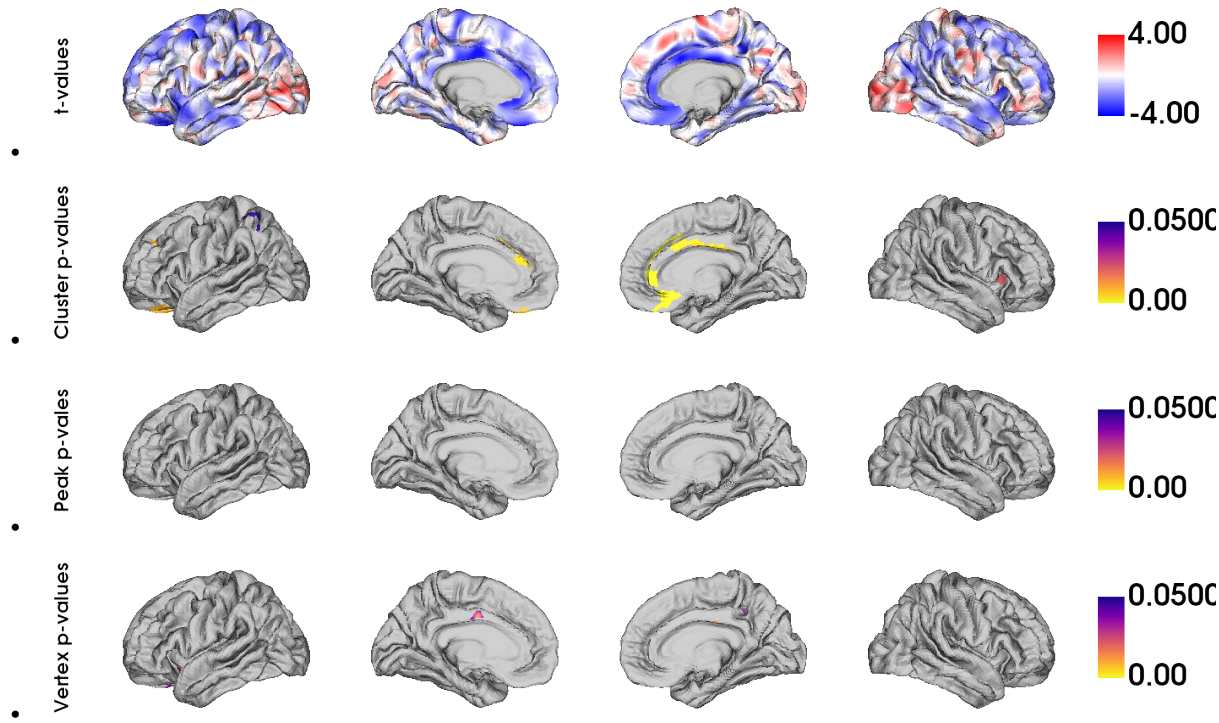
```
    intercept  AGE_AT_SCAN
0           1    -0.579678
1           1    -0.812116
2           1     0.117636
3           1     0.466293
4           1    -0.114802
..        ...          ...
77          1    -0.231021
78          1     0.117636
79          1    -0.695897
80          1    -0.695897
81          1    -0.347240

[82 rows x 2 columns]
```

Now, imagine we have some cortical marker (e.g. cortical thickness) for each subject, and we want to evaluate whether this marker is different across the the lifespan. To do this, we can use the model we defined before, and a contrast in observations (here: age). Then we simply initialize an SLM model and fit it to the cortical thickness data.

```
from brainstat.stats.SLM import SLM

contrast_age = demographics.AGE_AT_SCAN
slm_age = SLM(
    model,
    contrast_age,
    surf="fsaverage5",
    mask=mask,
    correction=["fdr", "rft"],
    cluster_threshold=0.01,
)
slm_age.fit(thickness)
```

The resulting model, slm_age, will contain the t-statistic map, p-values derived with the requested corrections, and a myriad of other properties (see the API for more details). Lets plot the t-values and p-values on the surface. We'll do this a few times throughout the tutorial so lets define a function to do this.

```
def plot_slm_results(slm, plot_peak=False, plot_fdr=False):

    handles = [local_plot_hemispheres(slm.t, ["t-values"], (-4, 4), "bwr")]

    plot_pvalues = [np.copy(slm.P["pval"]["C"])]
    labels = ["Cluster p-values"]

    if plot_peak:
        plot_pvalues.append(np.copy(slm.P["pval"]["P"]))
        labels.append("Peak p-vales")

    if plot_fdr:
        plot_pvalues.append(np.copy(slm.Q))
        labels.append("Vertex p-values")

    [np.place(x, np.logical_or(x > 0.05, ~mask), np.nan) for x in plot_pvalues]

    for i in range(len(plot_pvalues)):
        handles.append(
            local_plot_hemispheres(plot_pvalues[i], [labels[i]], (0, 0.05), "plasma_r
↪")
        )

    return handles


plot_slm_results(slm_age, plot_peak=True, plot_fdr=True)
```



Out:

```
[<IPython.core.display.Image object>, <IPython.core.display.Image object>, <IPython.
↪core.display.Image object>, <IPython.core.display.Image object>]
```

Only clusters are significant, and not peaks. This suggests that the age effect covers large regions, rather than local foci. Furthermore, at the vertexwise level we only find a small group of significant vertices in the left cingulate cortex. Lets have a closer look at the clusters and their peaks. The data on clusters are stored in tables inside BrainStatModel.P.clus and information on the peaks is stored in BrainStatModel.P.peak. If a two-tailed test is run (BrainStat defaults to two-tailed), a table is returned for each tail. The first table uses the contrast as provided, the second table uses the inverse contrast. If a one-tailed test is performed, then only a single table is returned. Lets print the first 15 rows of the inverted contrast cluster table.

```
print(slm_age.P["clus"][1])
```

Out:

```
    clusid  nverts    resels         P
0        1   141.0  6.283315  0.000033
1        2    82.0  3.994467  0.001858
2        3    69.0  3.871711  0.002362
3        4    61.0  3.670485  0.003517
4        5    82.0  3.652319  0.003648
..     ...     ...       ...       ...
73      74     1.0  0.050811  1.000000
74      75     1.0  0.043958  1.000000
75      76     1.0  0.039022  1.000000
76      77     1.0  0.032002  1.000000
77      78     1.0  0.019503  1.000000

[78 rows x 4 columns]
```

Here, we see that cluster 1 contains 373 vertices. Clusters are sorted by p-value; later clusters will generally be smaller and have higher p-values. Lets now have a look at the peaks within these clusters.

```
print(slm_age.P["peak"][1])
```

Out:

```
            t  vertid  clusid          P               yeo7
0    5.695420   18720      11   0.001248  Ventral Attention
1    5.164823    5430      12   0.009035             Limbic
2    4.855500   16911       6   0.027242  Ventral Attention
3    4.833974   19629       2   0.029335     Frontoparietal
4    4.628306   12603      14   0.059519       Default mode
..        ...     ...     ...        ...                ...
109  2.403000    2276      62  23.356468  Ventral Attention
110  2.394788    2185      74  23.709038       Default mode
111  2.389922   14687      76  23.918494       Default mode
112  2.382012    6087      64  24.258914       Default mode
113  2.375295    3243      72  24.548027       Default mode

[114 rows x 5 columns]
```

Within cluster 1, we are able to detect several peaks. The peak with the highest t-statistic (t=4.3972) occurs at vertex 19629, which is inside the frontoparietal network as defined by the Yeo-7 networks. Note that the Yeo network membership is only provided if the surface is specified as a template name as we did here. For custom surfaces, or pre-loaded surfaces (as we will use below) this column is omitted.

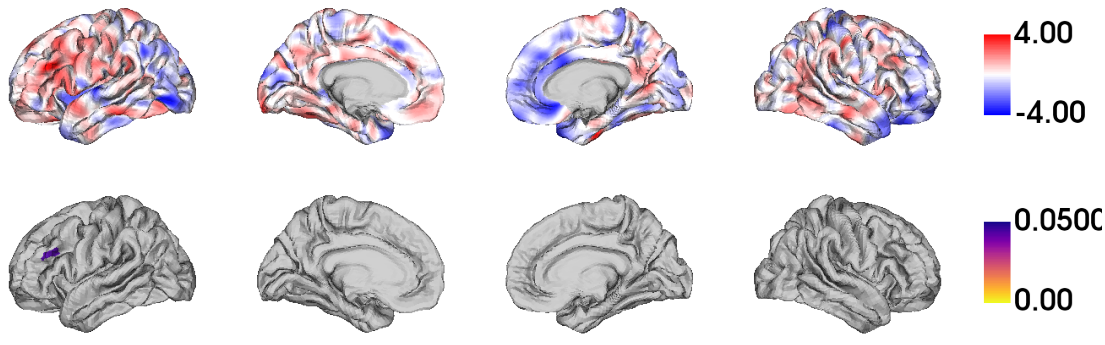
Interaction effects models

Similarly to age, we can also test for the effect of sex on cortical thickness.

```
term_sex = FixedEffect(demographics.SEX)
model_sex = term_sex
contrast_sex = (demographics.SEX == "M").astype(int) - (demographics.SEX == "F").
↪astype(
    int
)
```

Next we will rerun the model and see if our results change.

```
slm_sex = SLM(
    model_sex,
    contrast_sex,
    surf=pial_combined,
    mask=mask,
    correction=["fdr", "rft"],
    two_tailed=False,
    cluster_threshold=0.01,
)
slm_sex.fit(thickness)
plot_slm_results(slm_sex)
```



Out:

```
[<IPython.core.display.Image object>, <IPython.core.display.Image object>]
```

Here, we find few significant effects of sex on cortical thickness. However, as we've already established, age has an effect on cortical thickness. So we may want to correct for this effect before evaluating whether sex has an effect on cortical thickenss. Lets make a new model that includes the effect of age.

```
model_sexage = term_age + term_sex
```

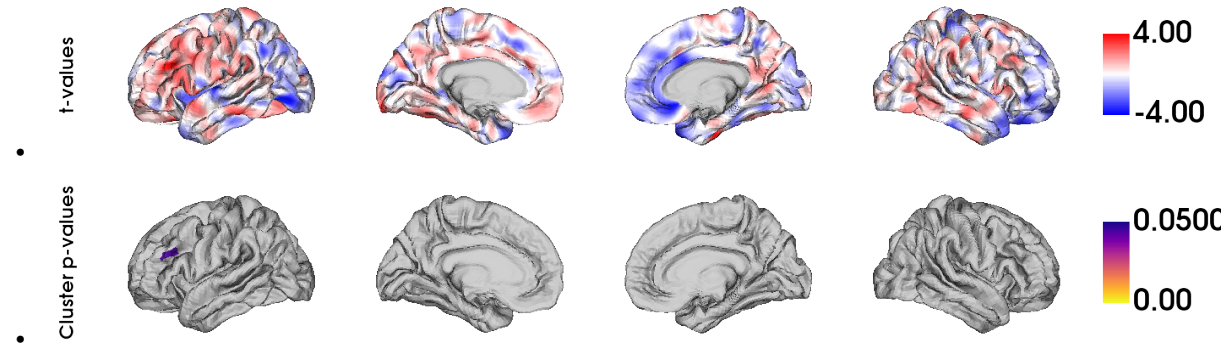
Next we will rerun the model and see if our results change.

```
slm_sexage = SLM(
    model_sexage,
    contrast_sex,
    surf=pial_combined,
    mask=mask,
    correction=["fdr", "rft"],
    two_tailed=False,
```

(continues on next page)

(continued from previous page)

```
    cluster_threshold=0.01,
)
slm_sexage.fit(thickness)
plot_slm_results(slm_sexage)
```

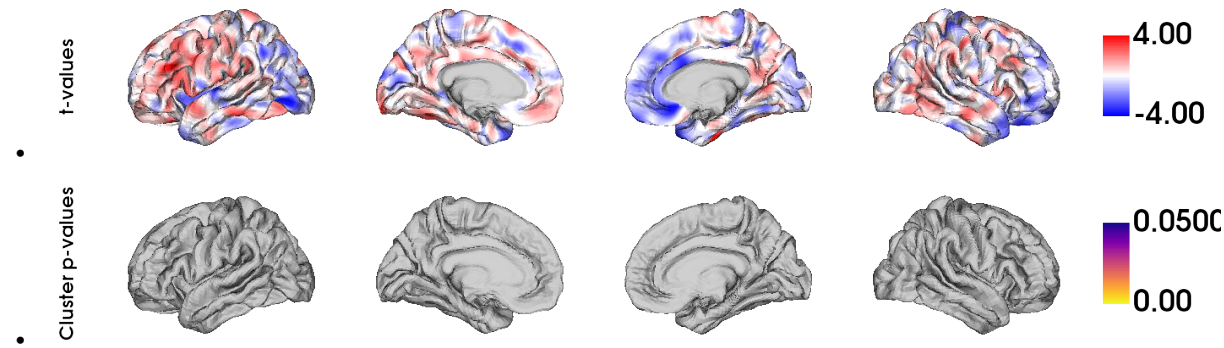


Out:

```
[<IPython.core.display.Image object>, <IPython.core.display.Image object>]
```

After accounting for the effect of age, we still find only one significant cluster of effect of sex on cortical thickness. However, it could be that age affects men and women differently. To account for this, we could include an interaction effect into the model. Lets run the model again with an interaction effect.

```
# Effect of sex on cortical thickness.
model_sexage_int = term_age + term_sex + term_age * term_sex

slm_sexage_int = SLM(
    model_sexage_int,
    contrast_sex,
    surf=pial_combined,
    mask=mask,
    correction=["rft"],
    cluster_threshold=0.01,
)
slm_sexage_int.fit(thickness)
plot_slm_results(slm_sexage_int)
```



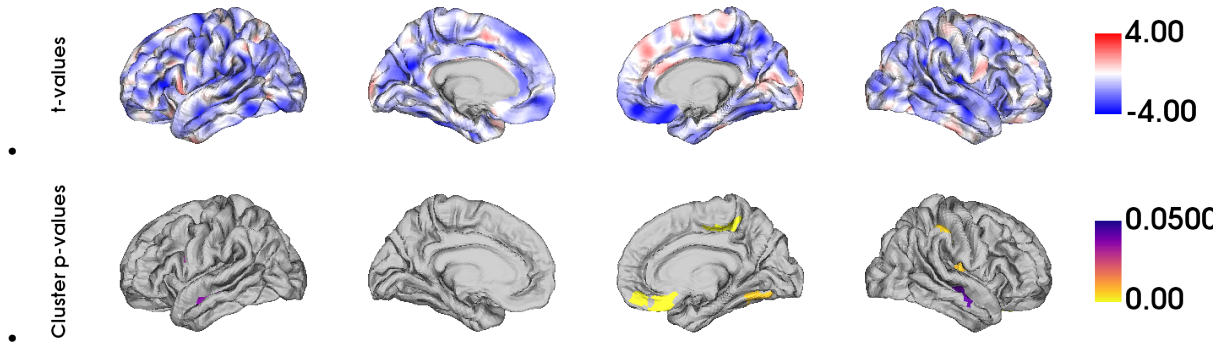
Out:

```
[<IPython.core.display.Image object>, <IPython.core.display.Image object>]
```

After including the interaction effect, we no significant effects of sex on cortical thickness in several clusters.

We could also look at whether the cortex of men and women changes differently with age by comparing their interaction effects.

```
# Effect of age on cortical thickness for the healthy group.
contrast_sex_int = demographics.AGE_AT_SCAN * (
    demographics.SEX == "M"
) - demographics.AGE_AT_SCAN * (demographics.SEX == "F")

slm_sex_int = SLM(
    model_sexage_int,
    contrast_sex_int,
    surf=pial_combined,
    mask=mask,
    correction=["rft"],
    cluster_threshold=0.01,
)
slm_sex_int.fit(thickness)
plot_slm_results(slm_sex_int)
```



Out:

```
[<IPython.core.display.Image object>, <IPython.core.display.Image object>]
```

Indeed, it appears that the interaction effect between sex and age is quite different across men and women, with stronger effects occuring in women.

One-tailed Test

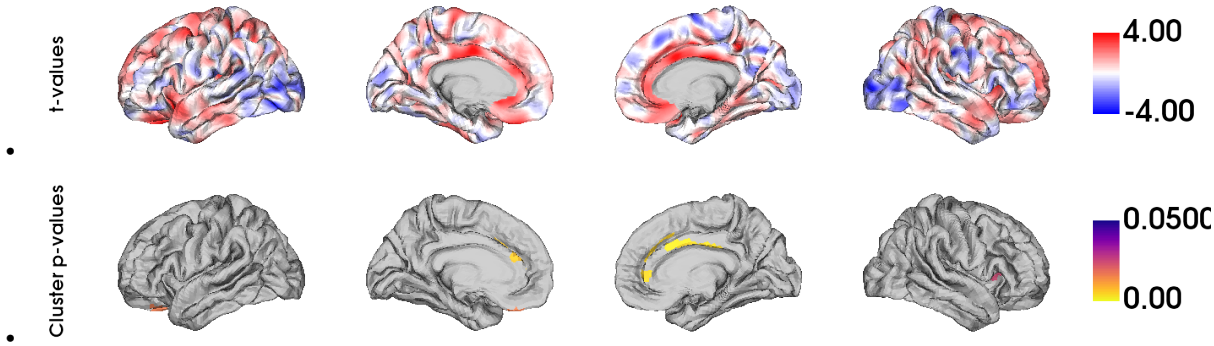
Imagine that, based on prior research, we hypothesize that men have higher cortical thickness than women. In that case, we could run this same model with a one-tailed test, rather than a two-tailed test. By default BrainStat uses a two-tailed test. If you want to get a one-tailed test, simply specify it in the SLM model initialization with 'two_tailed', false. Note that the one-tailed test will test for the significance of positive t-values. If you want to test for the significance of negative t-values, simply change the sign of the contrast. We may hypothesize based on prior research that cortical thickness decreases with age, so we could specify this as follows. Note the minus in front of contrast_age to test for decreasing thickness with age.

```
from brainstat.stats.SLM import SLM

slm_onetailed = SLM(
    model_sexage_int,
    -contrast_age,
```

(continues on next page)

(continued from previous page)

```
    surf=pial_combined,
    mask=mask,
    correction=["rft"],
    cluster_threshold=0.01,
    two_tailed=False,
)
slm_onetailed.fit(thickness)
plot_slm_results(slm_onetailed)
```



Out:

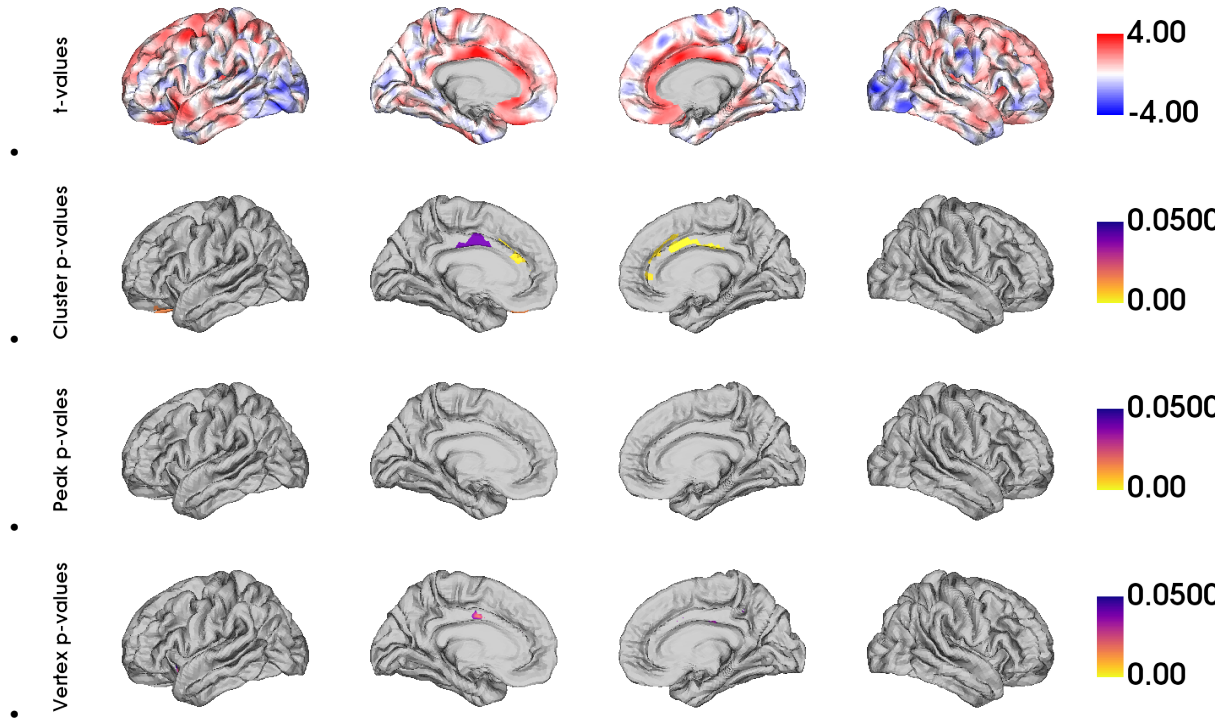
```
[<IPython.core.display.Image object>, <IPython.core.display.Image object>]
```

Notice the additional clusters that we find when using a one-tailed test.

Mixed Effects Models

So far, we've considered multiple visits of the same subject as two separate, independent measurements. Clearly, however, such measurements are not independent of each other. To account for this, we could add subject ID as a random effect. Lets do this and test the effect of age on cortical thickness again.

```
from brainstat.stats.terms import MixedEffect

term_subject = MixedEffect(demographics.SUB_ID)

model_mixed = term_age + term_sex + term_age * term_sex + term_subject

slm_mixed = SLM(
    model_mixed,
    -contrast_age,
    surf=pial_combined,
    mask=mask,
    correction=["fdr", "rft"],
    cluster_threshold=0.01,
    two_tailed=False,
)
slm_mixed.fit(thickness)
plot_slm_results(slm_mixed, True, True)
```



Out:

```
[<IPython.core.display.Image object>, <IPython.core.display.Image object>, <IPython.
↪core.display.Image object>, <IPython.core.display.Image object>]
```

Compared to our first age model, we find fewer and smaller clusters, indicating that by not accounting for the repeated measures structure of the data we were overestimating the significance of effects.

That concludes the basic usage of the BrainStat for statistical models. In the next tutorial we'll show you how to use the context decoding module.

Total running time of the script: (0 minutes 17.641 seconds)

Tutorial 02: Context Decoding

In this tutorial you will learn about the context decoding tools included with BrainStat. The context decoding module consists of three parts: genetic decoding, meta-analytic decoding and histological comparisons. Before we start, lets run a linear model testing for the effects of age on cortical thickness as we did in Tutorial 1. We'll use the results of this model later in this tutorial.

```
from brainstat.datasets import fetch_mask, fetch_template_surface
from brainstat.stats.SLM import SLM
from brainstat.stats.terms import FixedEffect, MixedEffect
from brainstat.tutorial.utils import fetch_mics_data

thickness, demographics = fetch_mics_data()
mask = fetch_mask("fsaverage5")

term_age = FixedEffect(demographics.AGE_AT_SCAN)
term_sex = FixedEffect(demographics.SEX)
term_subject = MixedEffect(demographics.SUB_ID)
```

(continues on next page)

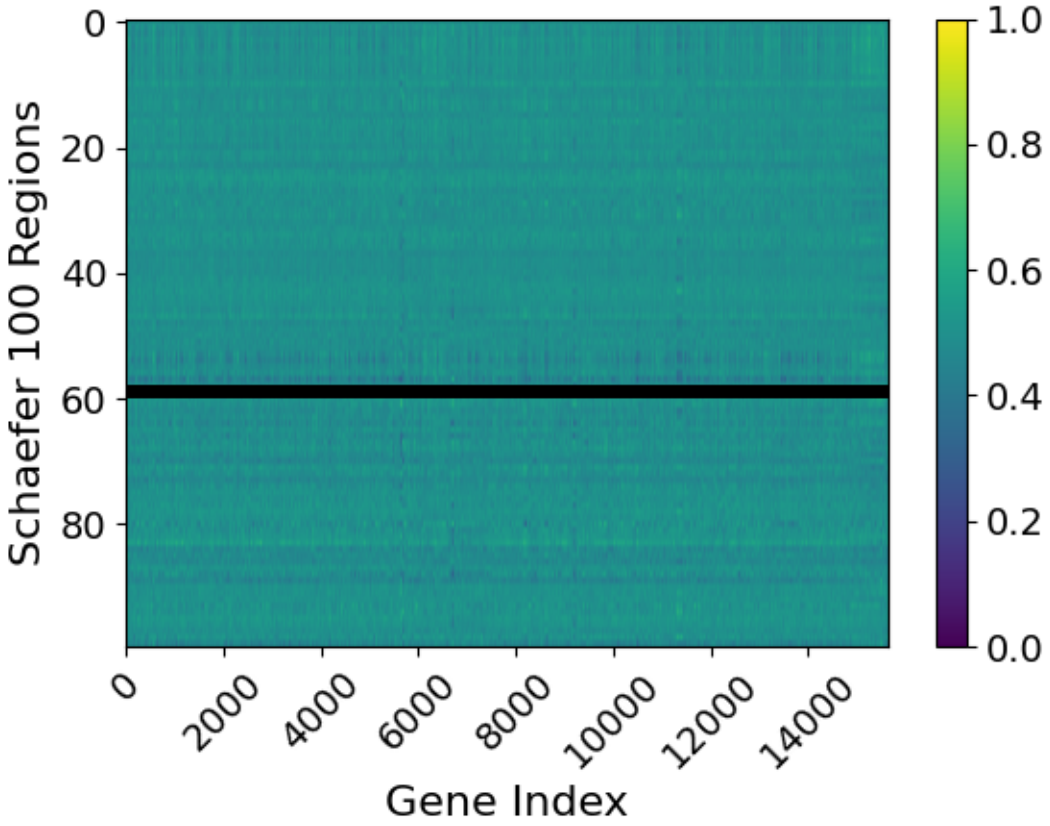
(continued from previous page)

```
model = term_age + term_sex + term_age * term_sex + term_subject

contrast_age = -model.mean.AGE_AT_SCAN
slm = SLM(
    model,
    contrast_age,
    surf="fsaverage5",
    mask=mask,
    correction=["fdr", "rft"],
    two_tailed=False,
    cluster_threshold=0.01,
)
slm.fit(thickness)
```

Genetics

For genetic decoding we use the Allen Human Brain Atlas through the abagen toolbox. Note that abagen only accepts parcellated data. Here is a minimal example of how we use abagen to get the genetic expression of the 100 regions of the Schaefer atlas and how to plot this expression to a matrix. Please note that downloading the dataset and running this analysis can take several minutes.

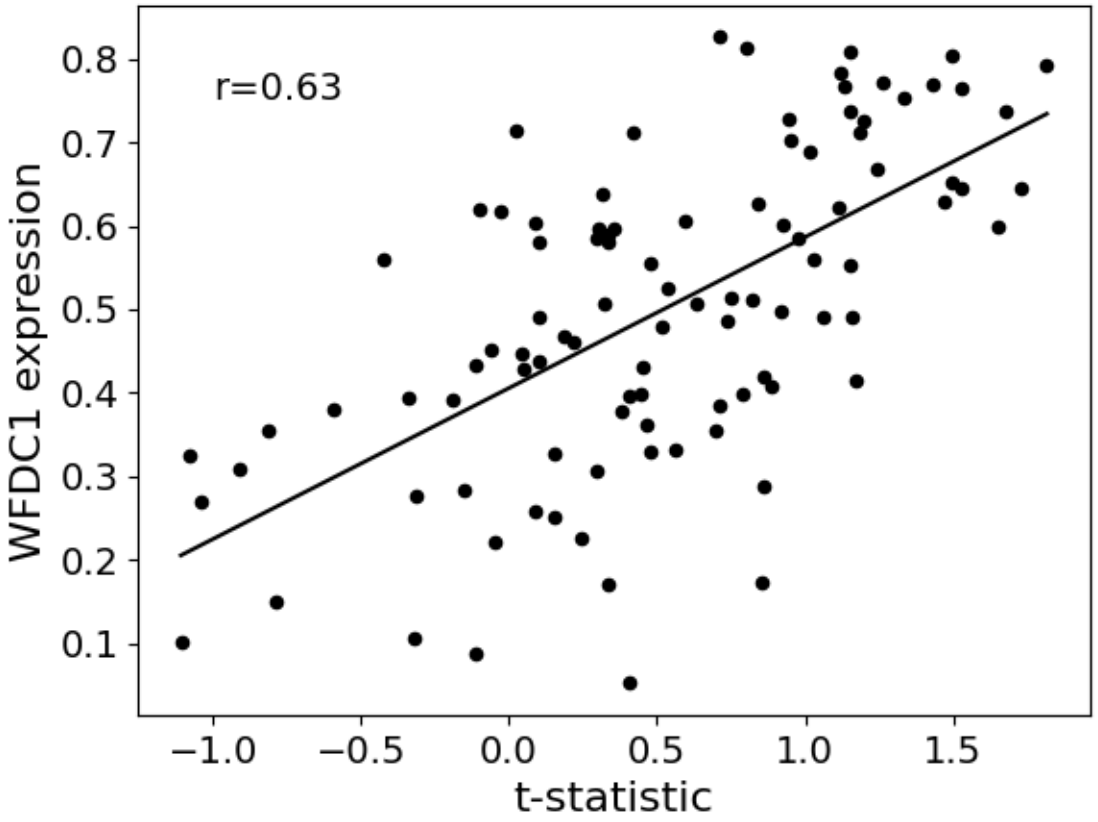
```
import copy

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from brainspace.utils.parcellation import reduce_by_labels
from matplotlib.cm import get_cmap

from brainstat.context.genetics import surface_genetic_expression
from brainstat.datasets import fetch_parcellation

# Get Schaefer-100 genetic expression.
schaefer_100_fs5 = fetch_parcellation("fsaverage5", "schaefer", 100)
surfaces = fetch_template_surface("fsaverage5", join=False)
expression = surface_genetic_expression(schaefer_100_fs5, surfaces, space="fsaverage")

# Plot Schaefer-100 genetic expression matrix.
colormap = copy.copy(get_cmap())
colormap.set_bad(color="black")
plt.imshow(expression, aspect="auto", cmap=colormap)
plt.colorbar().ax.tick_params(labelsize=14)
plt.xticks(fontsize=14, rotation=45)
plt.yticks(fontsize=14)
plt.xlabel("Gene Index", fontdict={"fontsize": 16})
plt.ylabel("Schaefer 100 Regions", fontdict={"fontsize": 16})
plt.gcf().subplots_adjust(bottom=0.2)
```



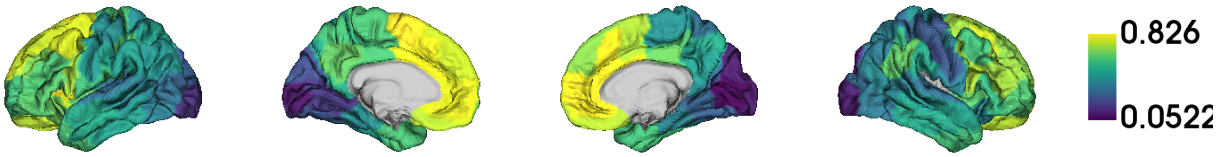
Expression is a pandas DataFrame which shows the genetic expression of genes within each region of the atlas. By default, the values will fall in the range [0, 1] where higher values represent higher expression. However, if you change the normalization function then this may change. Some regions may return NaN values for all genes. This occurs when there are no samples within this region across all donors. We've denoted this region with the black color in the matrix. By default, BrainStat uses all the default abagen parameters. If you wish to customize these parameters then the keyword arguments can be passed directly to *surface_genetic_expression*. For a full list of these arguments and their function please consult the abagen documentation.

Next, lets have a look at the correlation between one gene (WFDC1) and our t-statistic map. Lets also plot the expression of this gene to the surface.

```
# Plot correlation with WFDC1 gene
t_stat_schaefer_100 = reduce_by_labels(slm.t.flatten(), schaefer_100_fs5)[1:]

df = pd.DataFrame({"x": t_stat_schaefer_100, "y": expression["WFDC1"]})
df.dropna(inplace=True)
plt.scatter(df.x, df.y, s=20, c="k")
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.xlabel("t-statistic", fontdict={"fontsize": 16})
plt.ylabel("WFDC1 expression", fontdict={"fontsize": 16})
plt.plot(np.unique(df.x), np.poly1d(np.polyfit(df.x, df.y, 1))(np.unique(df.x)), "k")
plt.text(-1.0, 0.75, f"r={df.x.corr(df.y):.2f}", fontdict={"size": 14})
plt.show()
```




```
# Plot WFDC1 gene to the surface.
from brainspace.plotting.surface_plotting import plot_hemispheres
from brainspace.utils.parcellation import map_to_labels

vertexwise_WFDC1 = map_to_labels(
    expression["WFDC1"].to_numpy(),
    schaefer_100_fs5,
    mask=schaefer_100_fs5 != 0,
    fill=np.nan,
)

plot_hemispheres(
    surfaces[0],
    surfaces[1],
    vertexwise_WFDC1,
    color_bar=True,
    embed_nb=True,
    size=(1400, 200),
    zoom=1.45,
    nan_color=(0.7, 0.7, 0.7, 1),
    cb__labelTextProperty={"fontSize": 12},
)
```



Out:

```
/Users/reinder/opt/miniconda3/envs/python3.8/lib/python3.8/site-packages/brainspace/
↪plotting/base.py:287: UserWarning:

Interactive mode requires 'panel'. Setting 'interactive=False'


<IPython.core.display.Image object>
```

We find a small correlation. To test for significance we'll have to do some additional corrections, but more on that later.

Meta-Analytic

To perform meta-analytic decoding, BrainStat uses precomputed Neurosynth maps. Here we test which terms are most associated with a map of cortical thickness. A simple example analysis can be run as follows. The surface decoder interpolates the data from the surface to the voxels in the volume that are in between the two input surfaces. We'll decode the t-statistics derived with our model earlier. Note that downloading the dataset and running this analysis can take several minutes.

```
from brainstat.context.meta_analysis import meta_analytic_decoder

meta_analysis = meta_analytic_decoder("fsaverage5", slm.t.flatten())
print(meta_analysis)
```

Out:

```
                    Pearson's r
nucleus accumbens      0.207419
accumbens              0.207216
dorsal anterior        0.200371
dacc                   0.196472
ventral striatum       0.194027
...                         ...
selectivity           -0.225783
object recognition    -0.231140
v1                    -0.232876
lateral occipital     -0.233367
sighted               -0.250493

[3228 rows x 1 columns]
```

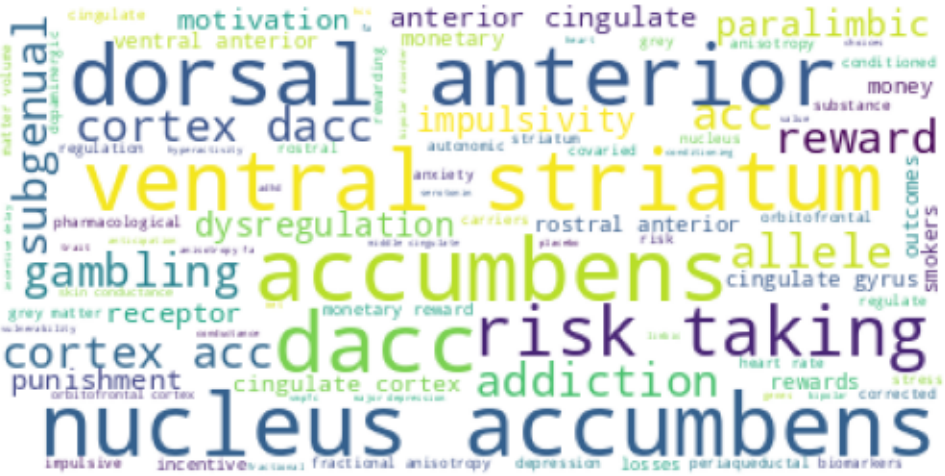
meta_analysis now contains a pandas.dataFrame with the correlation values for each requested feature. Next we could create a Wordcloud of the included terms, wherein larger words denote higher correlations.

```
from wordcloud import WordCloud

wc = WordCloud(background_color="white", random_state=0)
```

(continues on next page)

(continued from previous page)

```
wc.generate_from_frequencies(frequencies=meta_analysis.to_dict()["Pearson's r"])
plt.imshow(wc)
plt.axis("off")
plt.show()
```



If we broadly summarize, we see a lot of words related to language e.g., "language comprehension", "broca", "speaking", "speech production". Generally you'll also find several hits related to anatomy or clinical conditions. Depending on your research question, it may be more interesting to select only those terms related to cognition or some other subset.

Histological decoding

For histological decoding we use microstructural profile covariance gradients, as first shown by (Paquola et al, 2019, Plos Biology), computed from the BigBrain dataset. Firstly, lets download the MPC data, compute and plot its gradients, and correlate the first gradient with our t-statistic map.

```
from brainstat.context.histology import (
    compute_histology_gradients,
    compute_mpc,
    read_histology_profile,
)

# Run the analysis
```

(continues on next page)

(continued from previous page)

```
schaefer_400 = fetch_parcellation("fsaverage5", "schaefer", 400)
histology_profiles = read_histology_profile(template="fsaverage5")
mpc = compute_mpc(histology_profiles, labels=schaefer_400)
gradient_map = compute_histology_gradients(mpc, random_state=0)

# Bring parcellated data to vertex data.
vertexwise_gradient = map_to_labels(
    gradient_map.gradients_[:, 0],
    schaefer_400,
    mask=schaefer_400 != 0,
    fill=np.nan,
)

plot_hemispheres(
    surfaces[0],
    surfaces[1],
    vertexwise_gradient,
    embed_nb=True,
    nan_color=(0.7, 0.7, 0.7, 1),
    size=(1400, 200),
    zoom=1.45,
)
```

Out:

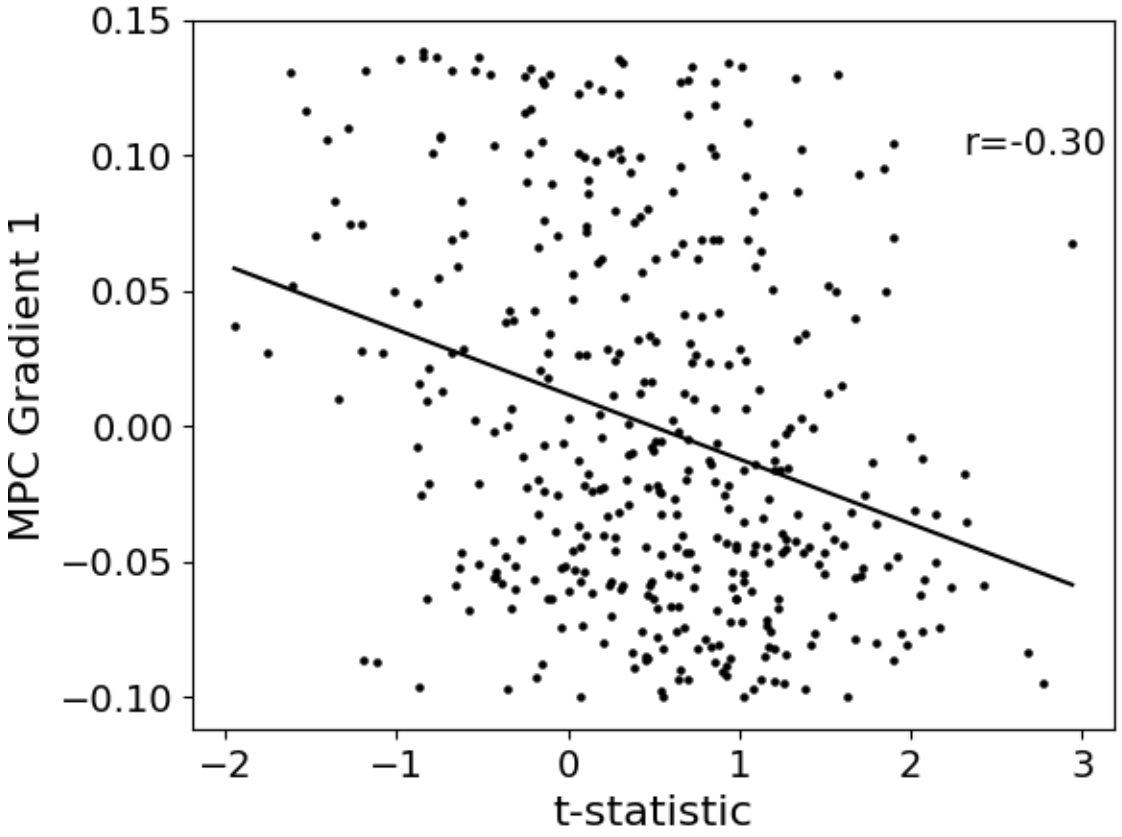
```
/Users/reinder/GitHub/BrainStat/brainstat/context/histology.py:105: RuntimeWarning:

divide by zero encountered in true_divide

/Users/reinder/GitHub/BrainStat/brainstat/context/histology.py:105: RuntimeWarning:

invalid value encountered in log

/Users/reinder/opt/miniconda3/envs/python3.8/lib/python3.8/site-packages/brainspace/
↪plotting/base.py:287: UserWarning:

Interactive mode requires 'panel'. Setting 'interactive=False'


<IPython.core.display.Image object>
```

```
# Plot the correlation between the t-stat
t_stat_schaefer_400 = reduce_by_labels(slm.t.flatten(), schaefer_400)[1:]
df = pd.DataFrame({"x": t_stat_schaefer_400, "y": gradient_map.gradients_[:, 0]})
df.dropna(inplace=True)
plt.scatter(df.x, df.y, s=5, c="k")
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.xlabel("t-statistic", fontdict={"fontsize": 16})
```

(continues on next page)

(continued from previous page)

```
plt.ylabel("MPC Gradient 1", fontdict={"fontsize": 16})
plt.plot(np.unique(df.x), np.poly1d(np.polyfit(df.x, df.y, 1))(np.unique(df.x)), "k")
plt.text(2.3, 0.1, f"r={df.x.corr(df.y):.2f}", fontdict={"size": 14})
plt.gcf().subplots_adjust(left=0.15)
plt.show()
```



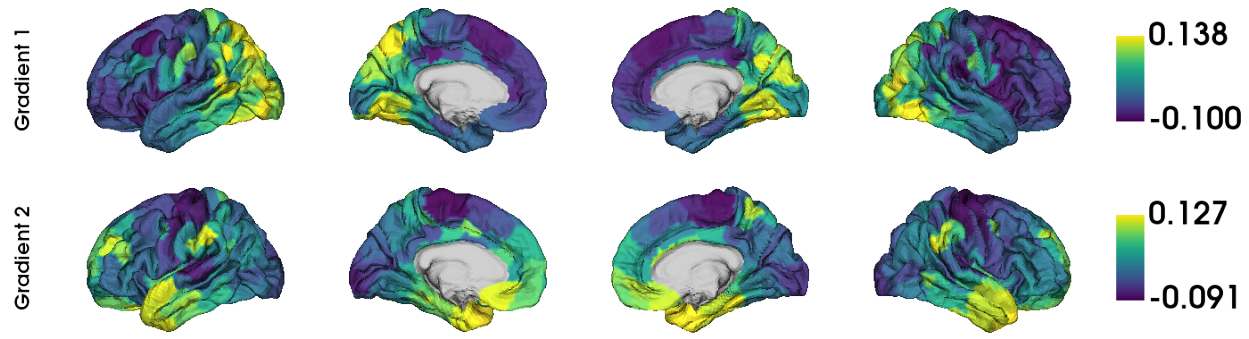
The variable histology_profiles now contains histological profiles sampled at 50 different depths across the cortex, mpc contains the covariance of these profiles, and gradient_map contains their gradients. We also see that the correlation between our t-statistic map and these gradients is not very high. Depending on your use-case, each of the three variables here could be of interest, but for purposes of this tutorial we'll plot the gradients to the surface with BrainSpace. For details on what the GradientMaps class (gradient_map) contains please consult the BrainSpace documentation.

```
from brainspace.utils.parcellation import map_to_labels

surfaces = fetch_template_surface("fsaverage5", join=False)

# Bring parcellated data to vertex data.
vertexwise_data = []
for i in range(0, 2):
    vertexwise_data.append(
        map_to_labels(
            gradient_map.gradients_[:, i],
            schaefer_400,
            mask=schaefer_400 != 0,
```

(continues on next page)

(continued from previous page)

```
            fill=np.nan,
        )
    )

# Plot to surface.
plot_hemispheres(
    surfaces[0],
    surfaces[1],
    vertexwise_data,
    embed_nb=True,
    label_text=["Gradient 1", "Gradient 2"],
    color_bar=True,
    size=(1400, 400),
    zoom=1.45,
    nan_color=(0.7, 0.7, 0.7, 1),
    cb__labelTextProperty={"fontSize": 12},
)
```



Out:

```
/Users/reinder/opt/miniconda3/envs/python3.8/lib/python3.8/site-packages/brainspace/
↪plotting/base.py:287: UserWarning:

Interactive mode requires 'panel'. Setting 'interactive=False'


<IPython.core.display.Image object>
```

Note that we no longer use the y-axis regression used in (Paquola et al, 2019, Plos Biology), as such the first gradient becomes an anterior-posterior gradient.

Resting-state contextualization

Lastly, BrainStat provides contextualization using resting-state fMRI markers: specifically, with the Yeo functional networks (Yeo et al., 2011, Journal of Neurophysiology), a clustering of resting-state connectivity, and the functional gradients (Margulies et al., 2016, PNAS), a lower dimensional manifold of resting-state connectivity.

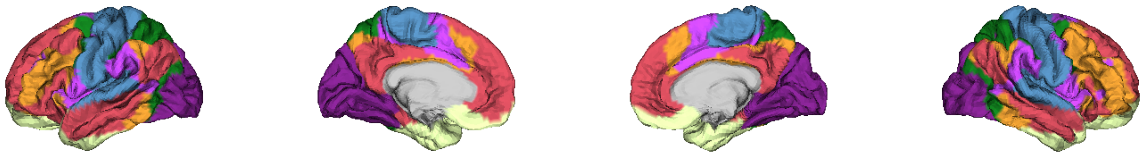
As an example, lets have a look at the the t-statistic map within the Yeo networks. We'll plot the Yeo networks as well as a barplot showing the mean and standard error of the mean within each network.

```
from matplotlib.colors import ListedColormap
```

(continues on next page)

(continued from previous page)

```
from brainstat.datasets import fetch_yeo_networks_metadata

yeo_networks = fetch_parcellation("fsaverage5", "yeo", 7)
network_names, yeo_colormap = fetch_yeo_networks_metadata(7)
yeo_colormap_gray = np.concatenate((np.array([[0.7, 0.7, 0.7]]), yeo_colormap))

plot_hemispheres(
    surfaces[0],
    surfaces[1],
    yeo_networks,
    embed_nb=True,
    cmap=ListedColormap(yeo_colormap_gray),
    nan_color=(0.7, 0.7, 0.7, 1),
    size=(1400, 200),
    zoom=1.45,
)
```



Out:

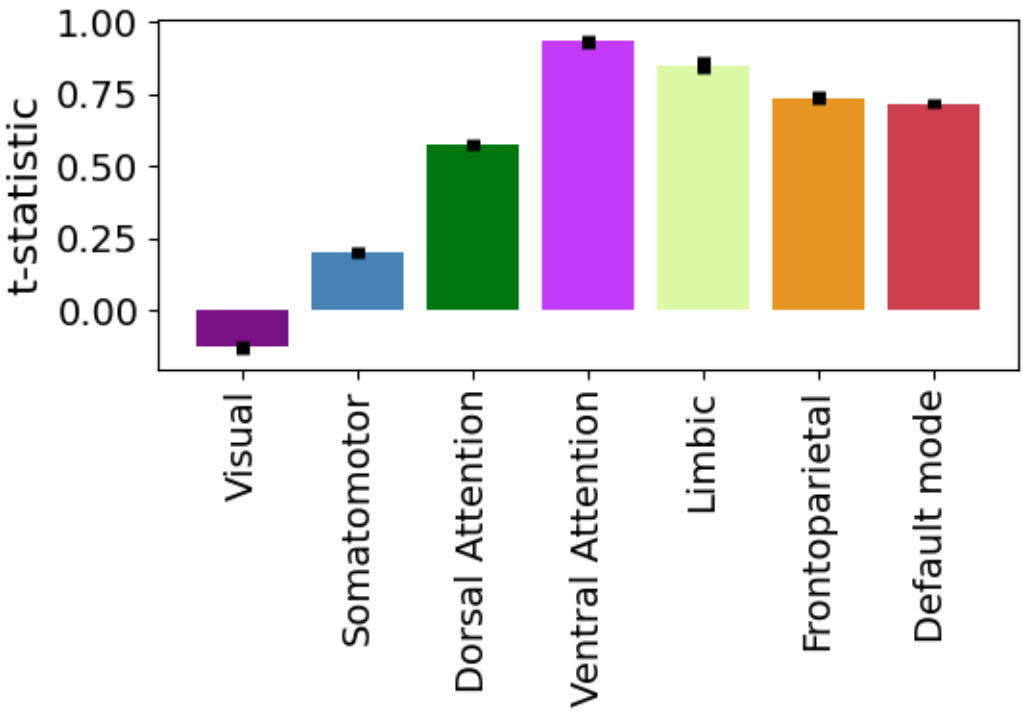
```
/Users/reinder/opt/miniconda3/envs/python3.8/lib/python3.8/site-packages/brainspace/
↪plotting/base.py:287: UserWarning:

Interactive mode requires 'panel'. Setting 'interactive=False'


<IPython.core.display.Image object>
```

```
import matplotlib.pyplot as plt
from scipy.stats import sem

from brainstat.context.resting import yeo_networks_associations

yeo_tstat_mean = yeo_networks_associations(slm.t.flatten(), "fsaverage5")
yeo_tstat_sem = yeo_networks_associations(
    slm.t.flatten(),
    "fsaverage5",
    reduction_operation=lambda x, y: sem(x, nan_policy="omit"),
)

plt.bar(
    np.arange(7),
    yeo_tstat_mean[:, 0],
    yerr=yeo_tstat_sem.flatten(),
    color=yeo_colormap,
    error_kw={"elinewidth": 5},
)
plt.xticks(np.arange(7), network_names, rotation=90, fontsize=14)
plt.yticks(fontsize=14)
plt.ylabel("t-statistic", fontdict={"fontsize": 16})
```

(continues on next page)

(continued from previous page)

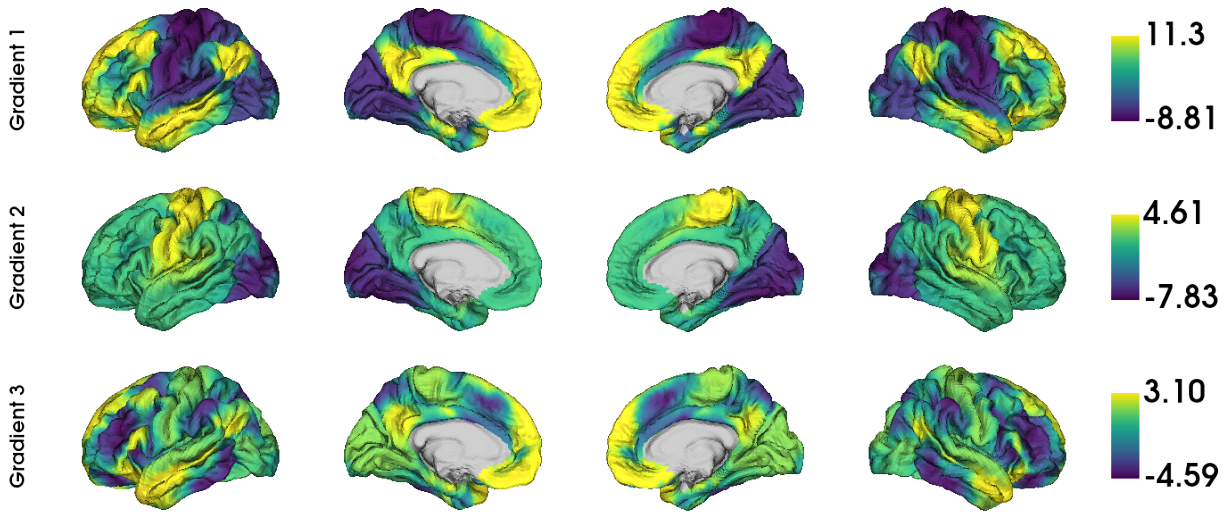
```
plt.gcf().subplots_adjust(left=0.2, bottom=0.5)
plt.show()
```



Across all networks, the mean t-statistic appears to be negative, with the most negative values in the dorsal attnetion and visual networks.

Lastly, lets plot the functional gradients and have a look at their correlation with our t-map.

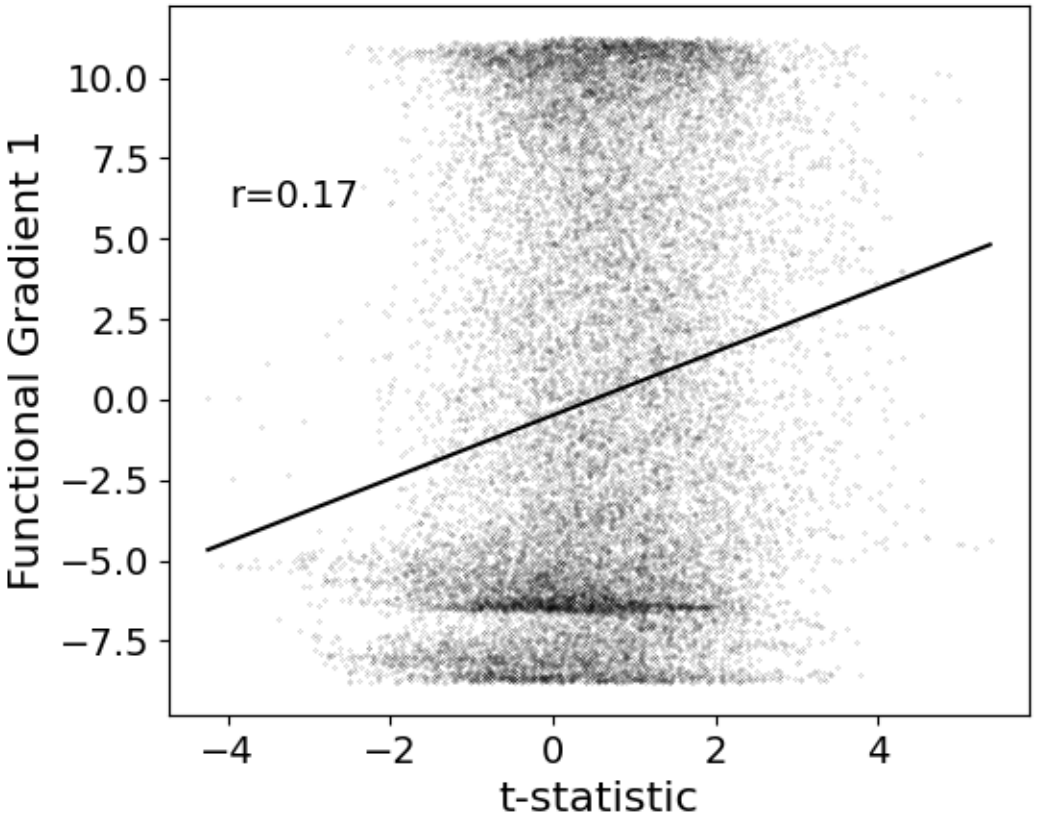
```
from brainstat.datasets import fetch_gradients

functional_gradients = fetch_gradients("fsaverage5", "margulies2016")


plot_hemispheres(
    surfaces[0],
    surfaces[1],
    functional_gradients[:, 0:3].T,
    color_bar=True,
    label_text=["Gradient 1", "Gradient 2", "Gradient 3"],
    embed_nb=True,
    size=(1400, 600),
    zoom=1.45,
    nan_color=(0.7, 0.7, 0.7, 1),
    cb__labelTextProperty={"fontSize": 12},
)
```



Out:

```
/Users/reinder/opt/miniconda3/envs/python3.8/lib/python3.8/site-packages/brainspace/
↪plotting/base.py:287: UserWarning:

Interactive mode requires 'panel'. Setting 'interactive=False'


<IPython.core.display.Image object>
```

```
df = pd.DataFrame({"x": slm.t.flatten(), "y": functional_gradients[:, 0]})
df.dropna(inplace=True)
plt.scatter(df.x, df.y, s=0.01, c="k")
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)
plt.xlabel("t-statistic", fontdict={"fontsize": 16})
plt.ylabel("Functional Gradient 1", fontdict={"fontsize": 16})
plt.plot(np.unique(df.x), np.poly1d(np.polyfit(df.x, df.y, 1))(np.unique(df.x)), "k")
plt.text(-4.0, 6, f"r={df.x.corr(df.y):.2f}", fontdict={"size": 14})
plt.gcf().subplots_adjust(left=0.2)
plt.show()
```



It seems the correlations are quite low. However, we'll need some more complex tests to assess statistical significance. There are many ways to compare these gradients to cortical markers. In general, we recommend using corrections for spatial autocorrelation which are implemented in BrainSpace. We'll show a correction with spin test in this tutorial; for other methods and further details please consult the BrainSpace tutorials.

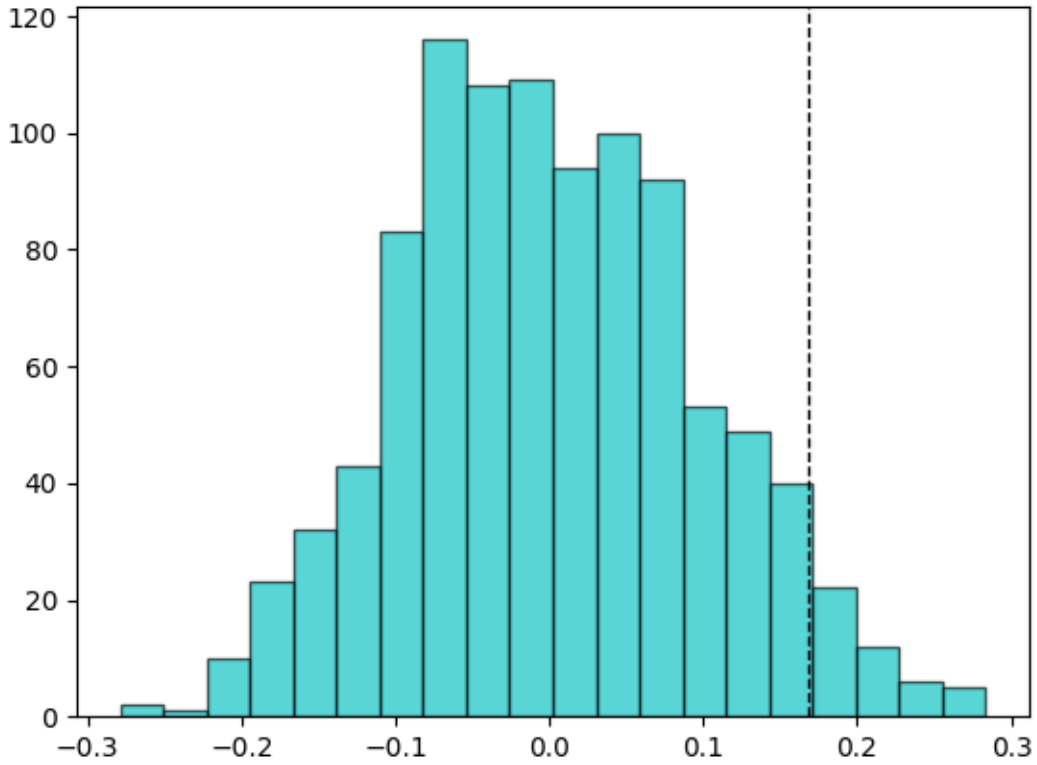
In a spin test we compare the empirical correlation between the gradient and the cortical marker to a distribution of correlations derived from data rotated across the cortical surface. The p-value then depends on the percentile of the empirical correlation within the permuted distribution.

```
from brainspace.null_models import SpinPermutations

sphere_left, sphere_right = fetch_template_surface(
    "fsaverage5", layer="sphere", join=False
)
tstat = slm.t.flatten()
tstat_left = tstat[: slm.t.size // 2]
tstat_right = tstat[slm.t.size // 2 :]

# Run spin test with 1000 permutations.
n_rep = 1000
sp = SpinPermutations(n_rep=n_rep, random_state=2021)
sp.fit(sphere_left, points_rh=sphere_right)
tstat_rotated = np.hstack(sp.randomize(tstat_left, tstat_right))

# Compute correlation for empirical and permuted data.
mask = ~np.isnan(functional_gradients[:, 0]) & ~np.isnan(tstat)
```

(continues on next page)

(continued from previous page)

```
r_empirical = np.corrcoef(functional_gradients[mask, 0], tstat[mask])[0, 1]
r_permuted = np.zeros(n_rep)
for i in range(n_rep):
    mask = ~np.isnan(functional_gradients[:, 0]) & ~np.isnan(tstat_rotated[i, :])
    r_permuted[i] = np.corrcoef(functional_gradients[mask, 0], tstat_rotated[i,␣
↪mask])[
        1:, 0
    ]

# Significance depends on whether we do a one-tailed or two-tailed test.
# If one-tailed it depends on in which direction the test is.
p_value_right_tailed = np.mean(r_empirical > r_permuted)
p_value_left_tailed = np.mean(r_empirical < r_permuted)
p_value_two_tailed = np.minimum(p_value_right_tailed, p_value_left_tailed) * 2
print(f"Two tailed p-value: {p_value_two_tailed}")

# Plot the permuted distribution of correlations.
plt.hist(r_permuted, bins=20, color="c", edgecolor="k", alpha=0.65)
plt.axvline(r_empirical, color="k", linestyle="dashed", linewidth=1)
plt.show()
```



Out:

```
Two tailed p-value: 0.094
```

As we can see from both the p-value as well as the histogram, wherein the dotted line denotes the empirical correlation, this correlation does not reach significance.

That concludes the tutorials of BrainStat. If anything is unclear, or if you think you've found a bug, please post it to the Issues page of our Github.

Happy BrainStating!

Total running time of the script: (4 minutes 12.410 seconds)

3.2.2 API

Neuroimaging statistics toolbox.

Modules

brainstat.context	BrainStat's context decoding module.
brainstat.datasets	Data included with BrainStat.
brainstat.mesh	Module for the handling of meshes and mesh data.
brainstat.stats	The statistics tools of BrainStat
brainstat.tests	Unit tests and their data generation.
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.histology	Histology context decoder
brainstat.context.meta_analysis	Meta-analytic decoding based on NiMARE
brainstat.context.resting	

brainstat.context.genetics

Genetic decoding using abagen.

Functions

surface_genetic_expression(labels[, ...])	Computes genetic expression of surface parcels.

brainstat.context.genetics.surface_genetic_expression

brainstat.context.genetics.**surface_genetic_expression**(*labels*, *surfaces=None*, *space=None*, ***, *atlas_info=None*, *ibf_threshold=0.5*, *probe_selection='diff_stability'*, *donor_probes='aggregate'*, *lr_mirror=None*, *missing=None*, *tolerance=2*, *sample_norm='srs'*, *gene_norm='srs'*, *norm_matched=True*, *norm_structures=False*, *region_agg='donors'*, *agg_metric='mean'*, *corrected_mni=True*, *reannotated=True*, *return_counts=False*, *return_donors=False*, *return_report=False*, *donors='all'*, *data_dir=None*, *verbose=0*, *n_proc=1*)

Computes genetic expression of surface parcels.

Parameters

- **labels** (*list-of-str or numpy.ndarray*) – List of paths to label files for the parcellation, or numpy array containing the pre-loaded labels
- **surfaces** (*list-of-image, optional*) – List of paths to surface files or preloaded surfaces. If not specified assumes that *labels* are on the *fsaverage5* surface. Default: None
- **space** (*{'fsaverage', 'fslr'}*) – What template space *surfaces* are aligned to. If not specified assumes that *labels* are on the *fsaverage5* surface. Default: None
- **details of the remaining parameters please consult the** (*For*) –
- **documentation. All its parameters bar "atlas"** (*abagen.get_expression_data()*) –
- **valid input parameters.** (*are*) –

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

Examples

```
>>> from brainstat.context.genetics import surface_genetic_expression
>>> from nilearn import datasets
>>> import numpy as np
```

```
>>> destrieux = datasets.fetch_atlas_surf_destrieux()
>>> labels = np.hstack((destrieux['map_left'], destrieux['map_right']))
>>> fsaverage = datasets.fetch_surf_fsaverage()
>>> surfaces = (fsaverage['pial_left'], fsaverage['pial_right'])
>>> expression = surface_genetic_expression(labels, surfaces,
...                                         space='fsaverage')
```

brainstat.context.histology

Histology context decoder

Functions

compute_histology_gradients(mpc[, kernel, ...])	Computes microstructural profile covariance gradients.
compute_mpc(profile, labels)	Computes MPC for given labels on a surface template.
download_histology_profiles([data_dir, ...])	Downloads BigBrain histology profiles.
partial_correlation(X, covar)	Runs a partial correlation whilst correcting for a covariate.
read_histology_profile([data_dir, template, ...])	Reads BigBrain histology profiles.

brainstat.context.histology.compute_histology_gradients

brainstat.context.histology.**compute_histology_gradients**(*mpc*, *kernel='normalized_angle'*, *approach='dm'*, *n_components=10*, *alignment=None*, *random_state=None*, *gamma=None*, *sparsity=0.9*, *reference=None*, *n_iter=10*)

Computes microstructural profile covariance gradients.

Parameters

- **mpc** (*numpy.ndarray*) – Microstructural profile covariance matrix.
- **kernel** (*str, optional*) – Kernel function to build the affinity matrix. Possible options: {'pearson', 'spearman', 'cosine', 'normalized_angle', 'gaussian'}. If callable, must receive a 2D array and return a 2D square array. If None, use input matrix. By default "normalized_angle".

- **approach** (*str, optional*) – Embedding approach. Can be 'pca' for Principal Component Analysis, 'le' for laplacian eigenmaps, or 'dm' for diffusion mapping, by default "dm".
- **n_components** (*int, optional*) – Number of components to return, by default 10.
- **alignment** (*str, None, optional*) – Alignment approach. Only used when two or more datasets are provided. Valid options are 'pa' for procrustes analysis and "joint" for joint embedding. If None, no alignment is peformed, by default None.
- **random_state** (*int, None, optional*) – Random state, by default None
- **gamma** (*float, None, optional*) – Inverse kernel width. Only used if kernel == "gaussian". If None, gamma=1/n_feat, by default None.
- **sparsity** (*float, optional*) – Proportion of smallest elements to zero-out for each row, by default 0.9.
- **reference** (*numpy.ndarray, optional*) – Initial reference for procrustes alignments. Only used when alignment == 'procrustes', by default None.
- **n_iter** (*int, optional*) – Number of iterations for Procrustes alignment, by default 10.

Returns *brainspace.gradient.gradient.GradientMaps* – BrainSpace gradient maps object.

brainstat.context.histology.compute_mpc

brainstat.context.histology.**compute_mpc**(*profile*, *labels*)

Computes MPC for given labels on a surface template.

Parameters

- **profile** (*numpy.ndarray*) – Histological profiles of size surface-by-vertex.
- **labels** (*numpy.ndarray*) – Labels of regions of interest. Use 0 to denote regions that will not be included.

Returns *numpy.ndarray* – Microstructural profile covariance.

brainstat.context.histology.download_histology_profiles

brainstat.context.histology.**download_histology_profiles**(*data_dir=None*, *template='fsaverage'*, *overwrite=False*)

Downloads BigBrain histology profiles.

Parameters

- **data_dir** (*str, pathlib,Path, None, optional*) – Path to the directory to store the data. If None, defaults to the home directory, by default None.
- **template** (*str, optional*) – Surface template. Currently allowed options are 'fsaverage' and 'fslr32k', by default 'fsaverage'.
- **overwrite** (*bool, optional*) – If true, existing data will be overwrriten, by default False.

Raises **KeyError** – Thrown if an invalid template is requested.

brainstat.context.histology.partial_correlation

brainstat.context.histology.**partial_correlation**(*X*, *covar*)

Runs a partial correlation whilst correcting for a covariate.

Parameters

- **X** (*ArrayLike*) – Two-dimensional array of the data to be correlated.
- **covar** (*numpy.ndarray*) – One-dimensional array of the covariate.

Returns *numpy.ndarray* – Partial correlation matrix.

brainstat.context.histology.read_histology_profile

brainstat.context.histology.**read_histology_profile**(*data_dir=None*, *template='fsaverage'*, *overwrite=False*)

Reads BigBrain histology profiles.

Parameters

- **data_dir** (*str, pathlib.Path, None, optional*) – Path to the data directory. If data is not found here then data will be downloaded. If None, data_dir is set to the home directory, by default None.
- **template** (*str, optional*) – Surface template. Currently allowed options are 'fsaverage' and 'fslr32k', by default 'fsaverage'.
- **overwrite** (*bool, optional*) – If true, existing data will be overwrriten, by default False.

Returns *numpy.ndarray* – Depth-by-vertex array of BigBrain intensities.

brainstat.context.meta_analysis

Meta-analytic decoding based on NiMARE

Functions

meta_analytic_decoder(template, stat_labels)	Meta-analytic decoding of surface maps using NeuroSynth or NeuroQuery.

brainstat.context.meta_analysis.meta_analytic_decoder

brainstat.context.meta_analysis.**meta_analytic_decoder**(*template*, *stat_labels*, *data_dir=None*)

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

Parameters

- **template** (*str*) – Path of a template volume file.
- **stat_labels** (*str, numpy.ndarray, sequence of str or numpy.ndarray*) – Path to a label file for the surfaces, numpy array containing the labels, or a

list containing multiple of the aforementioned.

- **data_dir** (*str, optional*) – The directory of the 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.

Returns *pandas.DataFrame* – Table with correlation values for each feature.

brainstat.context.resting

Functions

gradients_corr(data[, name, template, ...])	Comptues the correlation of the input data with the Margulies gradients.
yeo_networks_associations(data[, template, ...])	Computes association

brainstat.context.resting.gradients_corr

brainstat.context.resting.**gradients_corr**(*data*, *name='margulies2016'*, *template='fsaverage5'*, *data_dir=None*, *overwrite=False*)

Comptues the correlation of the input data with the Margulies gradients.

Parameters

- **data** (*ArrayLike*) – The data to be compared to the Margulies gradients. Data must be in the shape of vertices-by-features.
- **name** (*str, optional*) – Name of the gradients. Valid values are "margulies2016", defaults to "margulies2016".
- **template** (*str, optional*) – Name of the template surface. Valid values are "fsaverage5", "fsaverage7", "fslr32k", defaults to "fsaverage5".
- **data_dir** (*str, Path, optional*) – Path to the directory to store the Margulies gradient data files, by default $HOME_DIR/brainstat_data/functional_data.
- **overwrite** (*bool, optional*) – If true, overwrites existing files, by default False.

Returns *np.ndarray* – Correlations between the input data and the Margulies gradients.

brainstat.context.resting.yeo_networks_associations

brainstat.context.resting.**yeo_networks_associations**(*data*, *template='fsaverage5'*, *seven_networks=True*, *data_dir=None*, *reduction_operation=<function nanmean>*)

Computes association

Parameters

- **data** (*ArrayLike*) – Data to be summarized in the Yeo networks in a sample-by-feature format.

- **template** (*str, optional*) – Surface template. Valid values are "fsaverage5", "fsaverage", and "fslr32k", "civet41k", and "civet164k", by default "fsaverage5".
- **seven_networks** (*bool, optional*) – If true, uses the 7 network parcellation, otherwise uses the 17 network parcellation, by default True.
- **data_dir** (*str, Path, optional*) – Data directory to store the Yeo network files, by default $HOME_DIR/brainstat_data/parcellation_data.
- **reduction_operation** (*str, callable, optional*) – How to summarize data. If str, options are: {'min', 'max', 'sum', 'mean', 'median', 'mode', 'average'}. If callable, it should receive a 1D array of values, array of weights (or None) and return a scalar value. Default is 'mean'.

Returns *np.ndarray* – Summary statistic in the yeo networks.

brainstat.datasets

Data included with BrainStat.

Modules

brainstat.datasets.base	Load external datasets.

brainstat.datasets.base

Load external datasets.

Functions

fetch_gradients([template, name, data_dir, ...])	Fetch example gradients.
fetch_mask(template[, join, data_dir, overwrite])	Fetches midline masks.
fetch_parcellation(template, atlas, n_regions)	Loads the surface parcellation of a given atlas.
fetch_template_surface(template[, join, ...])	Loads surface templates.
fetch_yeo_networks_metadata(n)	Fetch Yeo networks metadata.

brainstat.datasets.base.fetch_gradients

brainstat.datasets.base.**fetch_gradients**(*template='fsaverage5'*, *name='margulies2016'*, *data_dir=None*, *overwrite=False*)

Fetch example gradients.

Parameters

- **template** (*str, optional*) – Name of the template surface. Valid values are "fsaverage5", "fsaverage", "fslr32k", defaults to "fsaverage5".
- **name** (*str*) – Name of the gradients. Valid values are "margulies2016", defaults to "margulies2016".
- **data_dir** (*str, Path, optional*) – Path to the directory to store the gradient data files, by default $HOME_DIR/brainstat_data/gradient_data.

- **overwrite** (*bool, optional*) – If true, overwrites existing files, by default False.

Returns *numpy.ndarray* – Vertex-by-gradient matrix.

brainstat.datasets.base.fetch_mask

brainstat.datasets.base.**fetch_mask**(*template*, *join=True*, *data_dir=None*, *overwrite=False*)

Fetches midline masks.

Parameters

- **template** (*str*) – Name of the surface template. Valid templates are: "fsaverage5", "fsaverage", "fslr32k", "civet41k", and "civet164k".
- **join** (*bool, optional*) – If true, returns a numpy array containing the mask. If false, returns a tuple containing the left and right hemispheric masks, respectively, by default True
- **data_dir** (*str, pathlib.Path, optional*) – Directory to save the data, by default $HOME_DIR/brainstat_data/surface_data.

Returns *numpy.ndarray or tuple of numpy.ndarray* – Midline mask, either as a single array or a tuple of a left and right hemispheric array.

brainstat.datasets.base.fetch_parcellation

brainstat.datasets.base.**fetch_parcellation**(*template*, *atlas*, *n_regions*, *join=True*, *seven_networks=True*, *data_dir=None*)

Loads the surface parcellation of a given atlas.

Parameters

- **template** (*str,*) – The surface template. Valid values are "fsaverage", "fsaverage5", "fsaverage6", "fslr32k", "civet41k", "civet164k", by default "fsaverage5".
- **atlas** (*str*) – Name of the atlas. Valid names are "cammoun", "glasser", "schaefer", "yeo".
- **n_regions** (*int*) – Number of regions of the requested atlas. Valid values for the cammoun atlas are 33, 60, 125, 250, 500. Valid values for the glasser atlas are 360. Valid values for the "schaefer" atlas are 100, 200, 300, 400, 500, 600, 800, 1000. Valid values for "yeo" are 7 and 17.
- **join** (*bool, optional*) – If true, returns parcellation as a single array, if false, returns an array per hemisphere, by default True.
- **seven_networks** (*bool, optional*) – If true, uses the 7 networks parcellation. Only used for the Schaefer atlas, by default True.
- **data_dir** (*str, pathlib.Path, optional*) – Directory to save the data, defaults to $HOME_DIR/brainstat_data/parcellation_data.

Returns *np.ndarray or tuple of np.npdarray* – Surface parcellation. If a tuple, then the first element is the left hemisphere.

brainstat.datasets.base.fetch_template_surface

brainstat.datasets.base.**fetch_template_surface**(*template*, *join=True*, *layer=None*, *data_dir=None*)

Loads surface templates.

Parameters

- **template** (*str*) – Name of the surface template. Valid values are "fslr32k", "fsaverage", "fsaverage3", "fsaverage4", "fsaverage5", "fsaverage6", "civet41k", "civet164k".
- **join** (*bool, optional*) – If true, returns surfaces as a single object, if false, returns an object per hemisphere, by default True.
- **layer** (*str, optional*) – Name of the cortical surface of interest. Valid values are "white", "smoothwm", "pial", "inflated", "sphere" for fsaverage surfaces; "midthickness", "inflated", "vinflated" for "fslr32k"; "mid", "white" for CIVET surfaces; and "sphere" for "civet41k". If None, defaults to "pial" or "midthickness", by default None.
- **data_dir** (*str, Path, optional*) – Directory to save the data, by default $HOME_DIR/brainstat_data/surface_data.

Returns *BSPolyData or tuple of BSPolyData* – Output surface(s). If a tuple, then the first element is the left hemisphere.

brainstat.datasets.base.fetch_yeo_networks_metadata

brainstat.datasets.base.**fetch_yeo_networks_metadata**(*n*)

Fetch Yeo networks metadata.

Parameters **n** (*int*) – Number of Yeo networks, either 7 or 17.

Returns

- *list of str* – Names of Yeo networks.
- *np.ndarray* – Colormap for the Yeo networks.

brainstat.mesh

Module for the handling of meshes and mesh data.

Modules

brainstat.mesh.data	Operations on data on a mesh.
brainstat.mesh.interpolate	Interpolations on a mesh.
brainstat.mesh.utils	Operations on meshes.

brainstat.mesh.data

Operations on data on a mesh.

Functions

mesh_smooth(Y, surf, FWHM)	Smooths surface data by repeatedly averaging over edges.

brainstat.mesh.data.mesh_smooth

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

Smooths surface data by repeatedly averaging over edges.

Parameters

- **Y** (*numpy.ndarray*) – Surface data of shape (n,v) or (n,v,k). v is the number of vertices, n is the number of observations, k is the number of variates.
- **surf** (*dict, BSPolyData*) – A dictionary with key 'tri' or 'lat', or a BSPolyData object of the surface. surf['tri'] is a numpy.ndarray of shape (t,3), t is the triangle indices, or surf['lat'] is a numpy.ndarray of shape (nx,ny,nz), where (nx,ny,nz) is the volume size, values are 1=in, 0=out.
- **FWHM** (*float*) – Gaussian smoothing filter in mesh units.

Returns *numpy.ndarray* – Smoothed surface data of shape (n,v) or (n,v,k).

brainstat.mesh.interpolate

Interpolations on a mesh.

Functions

combine_parcellations(files, output_file)	Combines multiple nifti files into one.
cortical_ribbon(pial_mesh, wm_mesh, nii[, ...])	Finds voxels inside of the cortical ribbon.
load_mesh_labels(label_file[, as_int])	Loads a .label.gii or .csv file.
multi_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.
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.combine_parcellations

brainstat.mesh.interpolate.**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.mesh.interpolate.cortical_ribbon

brainstat.mesh.interpolate.**cortical_ribbon**(*pial_mesh*, *wm_mesh*, *nii*, *mesh_distance=6*)
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** (*float, optional*) – Maximum distance from the cortical mesh at which the ribbon may occur. Used to reduce the search space, by default 6.

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

brainstat.mesh.interpolate.load_mesh_labels

brainstat.mesh.interpolate.**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.ndarray* – Labels in the file.

brainstat.mesh.interpolate.multi_surface_to_volume

brainstat.mesh.interpolate.**multi_surface_to_volume**(*pial*, *white*, *volume_template*, *output_file*, *labels*, *interpolation='nearest'*)

Interpolates multiple surfaces to the volume.

Parameters

- **pial** (*str, BSPolyData, list, tuple*) – Path of a pial surface file, BSPolyData of a pial surface or a list containing multiple of the aforementioned.
- **white** (*str, BSPolyData, list, tuple*) – Path of a white matter surface file, BSPolyData of a pial surface 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.
- **output_file** (*str*) – Path to the output file, must end in .nii or .nii.gz.
- **labels** (*str, numpy.ndarray, list, tuple*) – Path to a label file for the surfaces, numpy array containing the labels, or a list containing multiple of the aforementioned.
- **interpolation** (*str*) – Either 'nearest' for nearest neighbor interpolation, or 'linear' for trilinear interpolation, defaults to 'nearest'.

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.mesh.interpolate.read_surface_gz

brainstat.mesh.interpolate.**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.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.

- **interpolation** (*str, optional*) – Interpolation method. Can be either 'nearest' or 'linear'.

Returns *numpy.ndarray* – 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'*)

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'.

brainstat.mesh.utils

Operations on meshes.

Functions

mesh_edges(surf[, mask])	Converts the triangles or lattices of a mesh to edges.

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*, *data_dir=None*)

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*, *data_dir=None*)

Constructor for the SLM class.

Parameters

- **model** (brainstat.stats.terms.FixedEffect, brainstat.stats.terms.MixedEffect) – The linear model to be fitted of dimensions (observations, predictors).
- **contrast** (*array-like*) – Vector of contrasts in the observations.
- **surf** (*str, 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, or a string containing a template name accepted by fetch_template_surface, by default None.
- **mask** (*array-like, optional*) – A mask containing True for vertices to include in the analysis, by default None.
- **correction** (*str, Sequence, optional*) – String or sequence 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.
- **data_dir** (*str, pathlib.Path, optional*) – Path to the location to store BrainStat data files, defaults to $HOME_DIR/brainstat_data.

Methods

__init__(model, contrast[, surf, mask, ...])	Constructor for the SLM class.
fdr(**kwargs)	
fit(Y)	Fits the SLM model
linear_model(**kwargs)	
multiple_comparison_corrections(student_t_test)	Performs multiple comparisons corrections.
random_field_theory(**kwargs)	
t_test(**kwargs)	

Attributes

lat
surf
tri

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.

multiple_comparison_corrections(*student_t_test*)
Performs multiple comparisons corrections. If a (one-sided) student-t test was run, then make it two-tailed if requested.

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 FixedEffect, Series or DataFrame.
get_index(df)	Get index for column names of the form x{i}.
remove_duplicate_columns(df[, tol])	Remove duplicate columns.
remove_identical_columns(df1, df2)	Remove columns with duplicate names across dataframes.
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 FixedEffect, 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.remove_identical_columns

brainstat.stats.terms.**remove_identical_columns**(*df1*, *df2*)
Remove columns with duplicate names across dataframes.

Parameters

- **df1** (*DataFrame*) – Input dataframe from which to drop columns.
- **df2** (*DataFrame*) – Reference dataFrame

Returns *DataFrame* – df1 with columns appearing in df2 removed.

Raises **ValueError** – If there are columns with duplicate names but no duplicate values.

brainstat.stats.terms.to_df

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

Convert input to DataFrame.

Parameters

- **x** (*int, array-like FixedEffect*) – Input data.
- **n** (*int, optional*) – If input is a scalar, broadcast to column of *n* entries. Default is 1.
- **names** (*str, sequence 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

FixedEffect([x, names, add_intercept])	Build a term object for a linear model.
MixedEffect([ran, fix, name_ran, name_fix, ...])	Build a random term object for a linear model.

brainstat.stats.terms.FixedEffect

class brainstat.stats.terms.**FixedEffect**(*x=None*, *names=None*, *add_intercept=True*)

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.
- **add_intercept** (*bool, optional*) – If true, adds an intercept term. Defaults to True.

Variables

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

See also:

MixedEffect MixedEffect term

Examples

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

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

__init__ (*x=None*, *names=None*, *add_intercept=True*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([x, names, add_intercept])	Initialize self.

Attributes

is_empty
is_scalar
matrix
names
tolerance

brainstat.stats.terms.MixedEffect

class brainstat.stats.terms.**MixedEffect**(*ran=None*, *fix=None*, *name_ran=None*, *name_fix=None*, *ranisvar=False*, *add_intercept=True*, *add_identity=True*)

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, list, optional*) – Name(s) for the random term(s). 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** (FixedEffect) – FixedEffect for the mean.
- **variance** (FixedEffect) – FixedEffect for the variance.

See also:

FixedEffect FixedEffect object

Examples

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

```
>>> r2 = MixedEffect(np.arange(5), name_ran='r1')
>>> r2.mean.is_empty
True
>>> r2.variance.shape
(25, 2)
```

__init__(*ran=None*, *fix=None*, *name_ran=None*, *name_fix=None*, *ranisvar=False*, *add_intercept=True*, *add_identity=True*)
Initialize self. See help(type(self)) for accurate signature.

Methods

__init__([ran, fix, name_ran, name_fix, ...])	Initialize self.
broadcast_to(r1, r2)	
set_identity_last()	Sets the identity matrix column last.

Attributes

empty
shape

set_identity_last()
Sets the identity matrix column last.

Raises **ValueError** – Raised if "I" occurs more than once in the names.

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

Returns *numpy.ndarray* – 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** (*float*) – Starting number of the range.
- **stop** (*float*) – Stopping number of the range.
- **increment** (*float, optional*) – Increments of the range, defaults to 1.

Returns *numpy.array* – The requested numbers.

brainstat.stats.utils.interp1

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

Interpolation between datapoints.

Parameters

- **x** (*ArrayLike*) – x coordinates of training data.
- **y** (*ArrayLike*) – y coordinates of training data.

- **ix** (*ArrayLike*) – coordinates of the interpolated points.
- **str** (*kind*) – type of interpolation; see scipy.interpolate.interp1d for options.
- **int** – type of interpolation; see scipy.interpolate.interp1d for options.
- **optional** – 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.ndarray*) – 1D or 2D array
- **B** (*numpy.ndarray*) – 1D or 2D array
- **rows** (*bool, optional*) – If true test for row correspondence rather than element correpondence.

Returns

- *bool* – Boolean of the same size as A denoting which elements (or rows) occur in B.
- *numpy.ndarray* – 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 *list* – 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** (*numpy.ndarray*) – Masked data.
- **mask** (*numpy.ndarray*) – 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.ndarray* – Unmasked data.

brainstat.tests

Unit tests and their data generation.

Modules

brainstat.tests.datagen_f	Data generation for f-test unit tests.
brainstat.tests.datagen_linear_model	Data generation for linear model unit tests.
brainstat.tests.datagen_mesh_edges	Data generation for mesh_edges unit tests.
brainstat.tests.datagen_peak_clus	Data generation for peak_clus unit tests.
brainstat.tests.datagen_stat_threshold	Data generation for stat_threshold unit tests.
brainstat.tests.datagen_t_test	Data generation for t-test unit tests.
brainstat.tests.test_f_test	Unit tests of f-test.
brainstat.tests.test_mesh_edges	Unit tests of mesh_edges.
brainstat.tests.test_mesh_smooth	Unit tests of mesh_smooth.
brainstat.tests.test_stat_threshold	Unit tests of stat_threshold.
brainstat.tests.test_terms	Tests the fixed and mixed effects classes.
brainstat.tests.testutil	Utilities for running tests and test data generation.

brainstat.tests.datagen_f

Data generation for f-test unit tests.

Functions

generate_f_test_out(slm1, slm2)	
generate_random_two_slms(I)	
generate_test_data()	
params2files(I, D, test_num)	Converts params to input/output files

brainstat.tests.datagen_f.generate_f_test_out

brainstat.tests.datagen_f.**generate_f_test_out**(*slm1*, *slm2*)

brainstat.tests.datagen_f.generate_random_two_slms

brainstat.tests.datagen_f.**generate_random_two_slms**(*I*)

brainstat.tests.datagen_f.generate_test_data

brainstat.tests.datagen_f.**generate_test_data**()

brainstat.tests.datagen_f.params2files

brainstat.tests.datagen_f.**params2files**(*I*, *D*, *test_num*)
Converts params to input/output files

brainstat.tests.datagen_linear_model

Data generation for linear model unit tests.

Functions

generate_test_data()	

brainstat.tests.datagen_linear_model.generate_test_data

brainstat.tests.datagen_linear_model.**generate_test_data**()

brainstat.tests.datagen_mesh_edges

Data generation for mesh_edges unit tests.

Functions

generate_data_test_mesh_edges()	
generate_random_mesh_edge_data(key_dim, finname)	Generate random test datasets.
get_meshedge_output(surf, foutname)	Runs mesh_edges and returns all relevant output.

brainstat.tests.datagen_mesh_edges.generate_data_test_mesh_edges

brainstat.tests.datagen_mesh_edges.**generate_data_test_mesh_edges**()

brainstat.tests.datagen_mesh_edges.generate_random_mesh_edge_data

brainstat.tests.datagen_mesh_edges.**generate_random_mesh_edge_data**(*key_dim*, *finname*, *key_name='tri'*, *key_dtype='float'*, *seed=0*)

Generate random test datasets.

brainstat.tests.datagen_mesh_edges.get_meshedge_output

brainstat.tests.datagen_mesh_edges.**get_meshedge_output**(*surf*, *foutname*)

Runs mesh_edges and returns all relevant output.

brainstat.tests.datagen_peak_clus

Data generation for peak_clus unit tests.

Functions

generate_peak_clus_out(slm, I)	
generate_random_slm(I)	
generate_test_data()	
params2files(I, D, test_num)	Converts params to input/output files

brainstat.tests.datagen_peak_clus.generate_peak_clus_out

brainstat.tests.datagen_peak_clus.**generate_peak_clus_out**(*slm*, *I*)

brainstat.tests.datagen_peak_clus.generate_random_slm

brainstat.tests.datagen_peak_clus.**generate_random_slm**(*I*)

brainstat.tests.datagen_peak_clus.generate_test_data

brainstat.tests.datagen_peak_clus.**generate_test_data**()

brainstat.tests.datagen_peak_clus.params2files

brainstat.tests.datagen_peak_clus.**params2files**(*I*, *D*, *test_num*)
Converts params to input/output files

brainstat.tests.datagen_stat_threshold

Data generation for stat_threshold unit tests.

Functions

generate_stat_threshold_out(I)	
generate_test_data()	
params2files(I, D, test_num)	Converts params to input/output files

brainstat.tests.datagen_stat_threshold.generate_stat_threshold_out

brainstat.tests.datagen_stat_threshold.**generate_stat_threshold_out**(*I*)

brainstat.tests.datagen_stat_threshold.generate_test_data

brainstat.tests.datagen_stat_threshold.**generate_test_data**()

brainstat.tests.datagen_stat_threshold.params2files

brainstat.tests.datagen_stat_threshold.**params2files**(*I*, *D*, *test_num*)
Converts params to input/output files

brainstat.tests.datagen_t_test

Data generation for t-test unit tests.

Functions

generate_test_data()

brainstat.tests.datagen_t_test.generate_test_data

brainstat.tests.datagen_t_test.**generate_test_data**()

brainstat.tests.test_f_test

Unit tests of 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()

continues on next page

Table 32 – continued from previous page

test_11()
test_12()
test_13()
test_14()
test_15()
test_16()

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_f_test.test_13

brainstat.tests.test_f_test.**test_13**()

brainstat.tests.test_f_test.test_14

brainstat.tests.test_f_test.**test_14**()

brainstat.tests.test_f_test.test_15

brainstat.tests.test_f_test.**test_15**()

brainstat.tests.test_f_test.test_16

brainstat.tests.test_f_test.**test_16**()

brainstat.tests.test_mesh_edges

Unit tests of mesh_edges.

Functions

dummy_test(infile, expfile)	
test_01()	
test_02()	
test_03()	
test_04()	

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_smooth

Unit tests of mesh_smooth.

Functions

dummy_test(infile, expfile)	
test_01()	

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_stat_threshold

Unit tests of stat_threshold.

Functions

dummy_test(infile, expfile)	
test_01()	
test_02()	

continues on next page

Table 35 – continued from previous page

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()
test_21()
test_22()
test_23()
test_24()
test_25()
test_26()
test_27()

continues on next page

Table 35 – continued from previous page

test_28()
test_29()
test_30()
test_31()
test_32()
test_33()
test_34()
test_35()
test_36()
test_37()
test_38()
test_39()
test_40()
test_41()
test_42()
test_43()
test_44()
test_45()
test_46()
test_47()
test_48()

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_stat_threshold.test_17

brainstat.tests.test_stat_threshold.**test_17**()

brainstat.tests.test_stat_threshold.test_18

brainstat.tests.test_stat_threshold.**test_18**()

brainstat.tests.test_stat_threshold.test_19

brainstat.tests.test_stat_threshold.**test_19**()

brainstat.tests.test_stat_threshold.test_20

brainstat.tests.test_stat_threshold.**test_20**()

brainstat.tests.test_stat_threshold.test_21

brainstat.tests.test_stat_threshold.**test_21**()

brainstat.tests.test_stat_threshold.test_22

brainstat.tests.test_stat_threshold.**test_22**()

brainstat.tests.test_stat_threshold.test_23

brainstat.tests.test_stat_threshold.**test_23**()

brainstat.tests.test_stat_threshold.test_24

brainstat.tests.test_stat_threshold.**test_24**()

brainstat.tests.test_stat_threshold.test_25

brainstat.tests.test_stat_threshold.**test_25**()

brainstat.tests.test_stat_threshold.test_26

brainstat.tests.test_stat_threshold.**test_26**()

brainstat.tests.test_stat_threshold.test_27

brainstat.tests.test_stat_threshold.**test_27**()

brainstat.tests.test_stat_threshold.test_28

brainstat.tests.test_stat_threshold.**test_28**()

brainstat.tests.test_stat_threshold.test_29

brainstat.tests.test_stat_threshold.**test_29**()

brainstat.tests.test_stat_threshold.test_30

brainstat.tests.test_stat_threshold.**test_30**()

brainstat.tests.test_stat_threshold.test_31

brainstat.tests.test_stat_threshold.**test_31**()

brainstat.tests.test_stat_threshold.test_32

brainstat.tests.test_stat_threshold.**test_32**()

brainstat.tests.test_stat_threshold.test_33

brainstat.tests.test_stat_threshold.**test_33**()

brainstat.tests.test_stat_threshold.test_34

brainstat.tests.test_stat_threshold.**test_34**()

brainstat.tests.test_stat_threshold.test_35

brainstat.tests.test_stat_threshold.**test_35**()

brainstat.tests.test_stat_threshold.test_36

brainstat.tests.test_stat_threshold.**test_36**()

brainstat.tests.test_stat_threshold.test_37

brainstat.tests.test_stat_threshold.**test_37**()

brainstat.tests.test_stat_threshold.test_38

brainstat.tests.test_stat_threshold.**test_38**()

brainstat.tests.test_stat_threshold.test_39

brainstat.tests.test_stat_threshold.**test_39**()

brainstat.tests.test_stat_threshold.test_40

brainstat.tests.test_stat_threshold.**test_40**()

brainstat.tests.test_stat_threshold.test_41

brainstat.tests.test_stat_threshold.**test_41**()

brainstat.tests.test_stat_threshold.test_42

brainstat.tests.test_stat_threshold.**test_42**()

brainstat.tests.test_stat_threshold.test_43

brainstat.tests.test_stat_threshold.**test_43**()

brainstat.tests.test_stat_threshold.test_44

brainstat.tests.test_stat_threshold.**test_44**()

brainstat.tests.test_stat_threshold.test_45

brainstat.tests.test_stat_threshold.**test_45**()

brainstat.tests.test_stat_threshold.test_46

brainstat.tests.test_stat_threshold.**test_46**()

brainstat.tests.test_stat_threshold.test_47

brainstat.tests.test_stat_threshold.**test_47**()

brainstat.tests.test_stat_threshold.test_48

brainstat.tests.test_stat_threshold.**test_48**()

brainstat.tests.test_terms

Tests the fixed and mixed effects classes.

Functions

as_variance(M)	

test_fixed_init()	Tests the initialization of the FixedEffect class.
test_fixed_overload()	Tests the overloads of the FixedEffect class.
test_identity_detection()	Tests that the identity matrix is correctly placed last.
test_mixed_init()	Tests the initialization of the MixedEffect class.
test_mixed_overload()	Tests the overloads of the MixedEffect class.

brainstat.tests.test_terms.as_variance

brainstat.tests.test_terms.**as_variance**(*M*)

brainstat.tests.test_terms.test_fixed_init

brainstat.tests.test_terms.**test_fixed_init**()
Tests the initialization of the FixedEffect class.

brainstat.tests.test_terms.test_fixed_overload

brainstat.tests.test_terms.**test_fixed_overload**()
Tests the overloads of the FixedEffect class.

brainstat.tests.test_terms.test_identity_detection

brainstat.tests.test_terms.**test_identity_detection**()
Tests that the identity matrix is correctly placed last.

brainstat.tests.test_terms.test_mixed_init

brainstat.tests.test_terms.**test_mixed_init**()
Tests the initialization of the MixedEffect class.

brainstat.tests.test_terms.test_mixed_overload

brainstat.tests.test_terms.**test_mixed_overload**()
Tests the overloads of the MixedEffect class.

brainstat.tests.testutil

Utilities for running tests and test data generation.

Functions

array2effect(A[, n_random])	Converts an input array to a set of effects.
copy_slm(slm)	Copies an SLM object.
datadir(filename)	Returns the path to a given file in the test data directory.
generate_random_data_model(n_observations, ...)	Generates random test data.
generate_slm(**kwargs)	Generates a SLM with the given attributes :param All attributes of SLM can be provided as keyword arguments.:
save_input_dict(params, basename, test_num)	Saves the input data.
save_slm(slm, basename, testnum[, input])	Saves an SLM object containing test data.
slm2dict(slm)	Converts an SLM to a dictionary.
slm2files(slm, basename, test_num)	Converts an SLM to its output files.

brainstat.tests.testutil.array2effect

brainstat.tests.testutil.**array2effect**(*A*, *n_random=0*)

Converts an input array to a set of effects.

Parameters

- **A** (*np.array*) – A samples-by-effects array.
- **n_random** (*int, optional*) – Number of random effects, by default 0. Random effects are selected from the first columns of A.

Returns *brainstat.stats.terms.FixedEffect, brainstat.stats.terms.MixedEffect* – The fixed/mixed effects.

brainstat.tests.testutil.copy_slm

brainstat.tests.testutil.**copy_slm**(*slm*)

Copies an SLM object. :param slm: SLM object. :type slm: brainstat.stats.SLM.SLM

Returns *brainstat.stats.SLM.SLM* – SLM object.

brainstat.tests.testutil.datadir

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

Returns the path to a given file in the test data directory.

Parameters **filename** (*str*) – Name of a file in the data directory.

Returns *str* – Full path to file in the data directory.

brainstat.tests.testutil.generate_random_data_model

brainstat.tests.testutil.**generate_random_data_model**(*n_observations*, *n_vertices*, *n_variates*, *n_predictors*)

Generates random test data.

Parameters

- **n_observations** (*int*) – Number of observations.
- **n_vertices** (*int*) – Number of vertices.
- **n_variates** (*int*) – Number of variates.
- **n_predictors** (*int*) – Number of predictors.
- **n_random** (*int*) – Number of random effects.

Returns

- *numpy.ndarray* – Random data.
- *numpy.ndarray* – Random model.

brainstat.tests.testutil.generate_slm

brainstat.tests.testutil.**generate_slm**(***kwargs*)

Generates a SLM with the given attributes :param All attributes of SLM can be provided as keyword arguments.:

Returns *brainstat.stats.SLM.SLM* – SLM object.

brainstat.tests.testutil.save_input_dict

brainstat.tests.testutil.**save_input_dict**(*params*, *basename*, *test_num*)

Saves the input data.

Parameters

- **params** (*dict*) – Parameters provided by the parameter grid.
- **basename** (*str*) – Tag to save the file with.
- **test_num** (*int*) – Number of the test.

brainstat.tests.testutil.save_slm

brainstat.tests.testutil.**save_slm**(*slm*, *basename*, *testnum*, *input=True*)

Saves an SLM object containing test data. :param slm: SLM object. :type slm: brainstat.stats.SLM.SLM :param basename: Name for the tested function. :type basename: str :param testnum: Test number. :type testnum: int :param input: If True, appends _IN to filename. If false appends _OUT. :type input: boolean, optional

Returns *brainstat.stats.SLM.SLM* – SLM object.

brainstat.tests.testutil.slm2dict

brainstat.tests.testutil.**slm2dict**(*slm*)

Converts an SLM to a dictionary. :param slm: SLM object. :type slm: brainstat.stats.SLM.SLM

Returns *dict* – Dictionary with keys equivalent to the attributes of the slm.

brainstat.tests.testutil.slm2files

brainstat.tests.testutil.**slm2files**(*slm*, *basename*, *test_num*)

Converts an SLM to its output files.

Parameters

- **slm** (*brainstat.stats.SLM*) – SLM object.
- **basename** (*str*) – Base name for the file.
- **test_num** (*int*) – Number of the test.

brainstat.tutorial

Functions required for the BrainStat Tutorials

Modules

brainstat.tutorial.utils

brainstat.tutorial.utils

Functions

fetch_abide_data([data_dir, sites, ...])	Fetches ABIDE cortical thickness data.
fetch_mics_data([data_dir, overwrite])	Fetches MICS cortical thickness data.

brainstat.tutorial.utils.fetch_abide_data

brainstat.tutorial.utils.**fetch_abide_data**(*data_dir=None*, *sites=None*, *keep_control=True*, *keep_patient=True*, *overwrite=False*, *min_rater_ok=3*)

Fetches ABIDE cortical thickness data.

Parameters

- **data_dir** (*str, pathlib.Path, optional*) – Path to store the MICS data, by default $HOME_DIR/brainstat_data/mics_data.
- **sites** (*list, tuple, optional*) – List of sites to include. If none, uses all sites, by default None.
- **keep_control** (*bool, optional*) – If true keeps control subjects, by default True.
- **keep_patient** (*bool, optional*) – If true keeps patient subjects, by default True.
- **overwrite** (*bool, optional*) – If true overwrites existing data, by default False.
- **min_rater_ok** (*int, optional*) – Minimum number of raters who approved the data, by default 3.

Returns

- *np.ndarray* – Subject-by-vertex cortical thickness data on fsaverage5.
- *pd.DataFrame* – Subject demographics.

brainstat.tutorial.utils.fetch_mics_data

brainstat.tutorial.utils.**fetch_mics_data**(*data_dir=None*, *overwrite=False*)

Fetches MICS cortical thickness data.

Parameters

- **data_dir** (*str, pathlib.Path, optional*) – Path to store the MICS data, by default $HOME_DIR/brainstat_data/mics_data.
- **overwrite** (*bool, optional*) – If true overwrites existing data, by default False

Returns

- *np.ndarray* – Subject-by-vertex cortical thickness data on fsaverage5.
- *pd.DataFrame* – Subject demographics.

3.3 MATLAB Index

3.3.1 MATLAB Tutorials

For MATLAB tutorials, we recommend viewing these either through the Examples tab on our FileExchange page, or by opening the files in MATLAB (`PACKAGE_DIRECTORY/tutorials`). Alternatively, we've also included copies of these live scripts in ReadTheDocs:

- matlab_tutorial1
- matlab_tutorial2

3.3.2 MATLAB API

Below are links to descriptions of the important MATLAB functions used in BrainStat.

Statistics

- matlab_FixedEffect
- matlab_MixedEffect
- matlab_SLM

Context Decoding

- matlab_compute_histology_gradients
- matlab_compute_mpc
- matlab_fetch_genetic_expression
- matlab_read_histology_profile
- matlab_surface_decoder

Datasets

- matlab_fetch_abide_data
- matlab_fetch_gradients
- matlab_fetch_mask
- matlab_fetch_parcellation
- matlab_fetch_template_surface
- matlab_fetch_yeo_networks_metadata

I/O

- matlab_read_surface_data
- matlab_read_surface
- matlab_read_volume
- matlab_write_surface
- matlab_write_volume

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

- Savoy Foundation for Epilepsy (to RV)
- Richard and Ann Sievers Award (to RV)
- 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

Y