BrainSMASH

Joshua B. Burt

Mar 10, 2021

CONTENTS

BrainSMASH (Brain Surrogate Maps with Autocorrelated Spatial Heterogeneity) is a Python-based computational platform for statistical testing of spatially autocorrelated brain maps. At the core of BrainSMASH is functionality to simulate surrogate brain maps with spatial autocorrelation that is matched to spatial autocorrelation in a target brain map. Additional utilities are provided for Connectome Workbench users who wish to analyze surface-based neuroimaging files.

CHAPTER

ONE

CONTENTS:

1.1 Installation Guide

Installing BrainSMASH requires:

- Python 3+
- numpy
- pandas
- nibabel
- matplotlib
- scikit-learn
- scipy

BrainSMASH was developed and tested using Python 3.7. Backwards-compatibility with other Python3 versions is expected but not guaranteed. All Python package version dependencies are handled automatically if installing BrainSMASH via `pip` (described below).

Note: Using `nibabel` versions >2.1.0 will break the code, though this incompatibility will be resolved in a future release.

1.1.1 Dependencies

In addition to the Python packages listed above, *if and only if* you wish to use the additional utilities provided for Connectome Workbench users, you must have Connectome Workbench installed with the `wb_command` executable locatable in your system PATH environment variable. For Mac users with the unzipped `workbench` directory (downloaded here) located in your `Applications` folder, this can be achieved by simply adding `export PATH="/Applications/workbench/bin_macosx64:$PATH"` to your `.bash_profile`.

Note: For non-login interactive shells (which does not include Terminal), you may also need to configure `PATH` in your `.bashrc` file, as described above. Alternatively, for a single session only, you may run `echo 'export PATH=$PATH:/Applications/workbench/bin_macosx64' >> ~/.bashrc`.

1.1.2 Installation

BrainSmash is most easily installed via the Python package manager `pip`:

```
pip install brainsmash
```

You may also clone and install the source files manually via the GitHub repository:

```
git clone https://github.com/murraylab/brainsmash.git
cd brainsmash
python setup.py install
```

If you're concerned about clashing dependencies, BrainSMASH can be installed in a new `conda` environment:

```
conda create -n brainsmash python=3.7
conda activate brainsmash
pip install brainsmash
```

1.2 Approach

In BrainSMASH, spatial autocorrelation (SA) in brain maps is operationalized through the construction of a variogram:

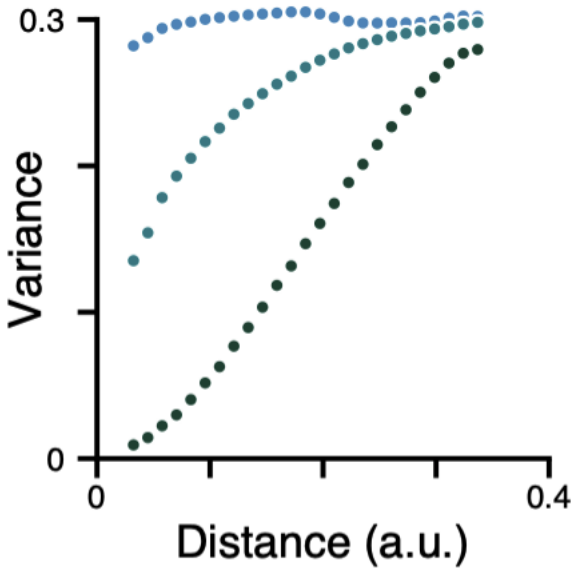


Fig. 1.1: Variograms provide summary measures of pairwise variation as a function of distance.

The variogram quantifies, as a function of distance d, the variance between all pairs of points spatially separated by d. Pure white noise, for example, which has equal variation across all spatial scales, has a flat variogram (i.e., no distance dependence). Brain maps with very little SA will therefore have a variogram which is nearly flat, like the blue curve above. In contrast, strongly autocorrelated brain maps exhibit less variation among spatially proximal regions – at small d – than among widely separated regions, and are therefore characterized by positive slopes in their variograms. The brain map which produced the dark green curve above is therefore more autocorrelated than the brain map which produced the cyan curve. **To generate SA-preserving surrogate brain maps, BrainSMASH produces random maps whose variograms are approximately matched to a target brain map's variogram.**

The figure below provides a schematic representation of the generative process implemented in BrainSMASH:

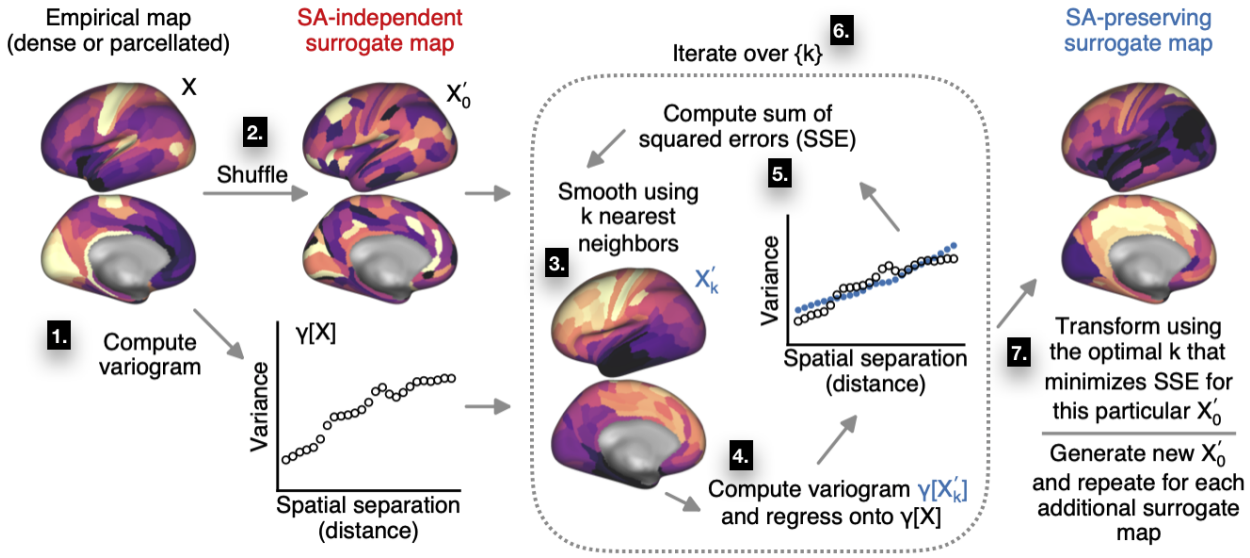


Fig. 1.2: Generating spatial autocorrelation-preserving surrogate maps.

The algorithm consists of the following steps:

1. The variogram for the target brain map is computed. This is the target variogram for the output surrogate maps.
2. The empirical map is randomly permuted, breaking its spatial structure and randomizing its topography.
3. Spatial autocorrelation among the samples is reintroduced by smoothing the permuted map with a distance-dependent kernel. (By default, BrainSMASH uses an exponentially decaying kernel, but other options are *available*.) Smoothing is performed using each region's k nearest neighboring regions. Different values of k correspond to differences in the characteristic length scale of the reintroduced SA.
4. The smoothed map's variogram is computed and then regressed onto the variogram for the empirical map. (The regression coefficients define the linear transformation of the smoothed map which approximately recovers the SA that is present in the target map.)
5. The goodness-of-fit is quantified by computing the sum of squared error (SSE) in the variogram fit.
6. Steps 3-5 are repeated, each time using a different number of nearest neighbors, k, to perform the spatial smoothing. (In BrainSMASH, k is parametrized as a fraction of the total number of regions; the set of values to iterate over may be specified by the user.)
7. The best value of k, which minimizes SSE, is used to generate a surrogate map whose SA is most closely matched to SA in the target map.

Steps 2-7 are repeated for each surrogate map. A more memory efficient implementation of the algorithm, which utilizes random sampling and memory-mapped arrays, is described *here* and in the preprint: in brief, steps 1 and 4 are performed on a random subset of brain areas, and the pairwise distance matrix is never loaded entirely into memory.

The distance matrix can in theory be constructed using any distance measure. Here and in the preprint, we use geodesic distance (i.e., distance along the cortical surface) for cortical surface brain maps, and three-dimensional Euclidean distance for subcortical volumetric brain maps.

1.3 Getting Started

1.3.1 Input data types

Using BrainSMASH requires specifying two inputs:

- A brain map, i.e. a one-dimensional scalar vector, and
- A distance matrix, containing a measure of distance between each pair of elements in the brain map

For illustration's sake, we will assume both required arguments have been written to disk as whitespace-separated text files `LeftParcelMyelin.txt` and `LeftParcelGeodesicDistmat.txt`. However, BrainSMASH can flexibly accommodate a variety of input types:

- Delimited `*.txt` files
- Neuroimaging files used by Connectome Workbench, including `*scalar.nii` files and `*.func.gii` metric files (assuming the files contain only a single brain map)
- Data and memory-mapped arrays written to `*.npy` files
- Numpy arrays and array-like objects

To follow along with the examples below, you can download our example data. Connectome Workbench users who wish to derive a distance matrix from a `*.surf.gii` file may want to begin *below*, as these functions take a long time to run (but thankfully only ever need to be run once).

1.3.2 Parcellated surrogate maps

For this example, we'll make the additional assumption that `LeftParcelMyelin.txt` contains myelin map values for 180 unilateral cortical parcels, and that `LeftParcelGeodesicDistmat.txt` is a 180x180 matrix containing the pairwise geodesic distances between parcels.

Because working with parcellated data is not computationally expensive, we'll import the `brainsmash.mapgen.base.Base` class (which does not utilize random sampling):

```
from brainsmash.mapgen.base import Base
brain_map_file = "LeftParcelMyelin.txt"  # use absolute paths if necessary!
dist_mat_file = "LeftParcelGeodesicDistmat.txt"
```

Note that if the two text files are not in the current directory, you'll need to include the absolute paths to the files in the variables defined above.

We'll create an instance of the class, passing our two files as arguments (implicitly using the default values for the optional keyword arguments):

```
base = Base(x=brain_map_file, D=dist_mat_file)
```

Surrogate maps can then be generated with a call to the class instance:

```
surrogates = base(n=1000)
```

where `surrogates` is a numpy array with shape `(1000,180)`. The empirical brain map and one of the surrogate maps are illustrated side-by-side below for comparison:

By construction, both maps exhibit the same degree of spatial autocorrelation in their values. However, notice that the empirical brain map has a distribution of values more skewed towards higher values, indicated by dark purple. If you wish to generate surrogate maps which preserve (identically) the distribution of values in the empirical map, use the keyword argument `resample` when instantiating the class:

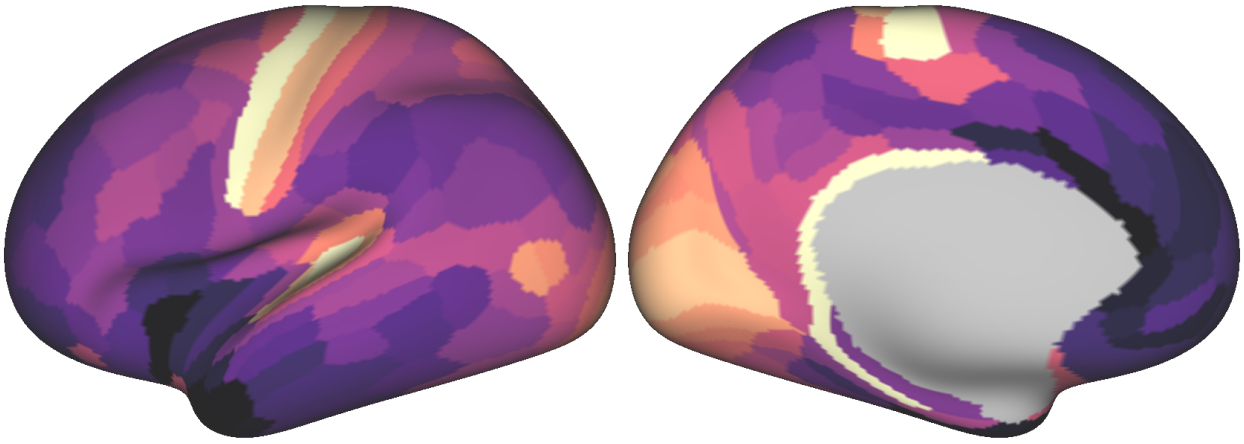


Fig. 1.3: The empirical T1w/T2w map.

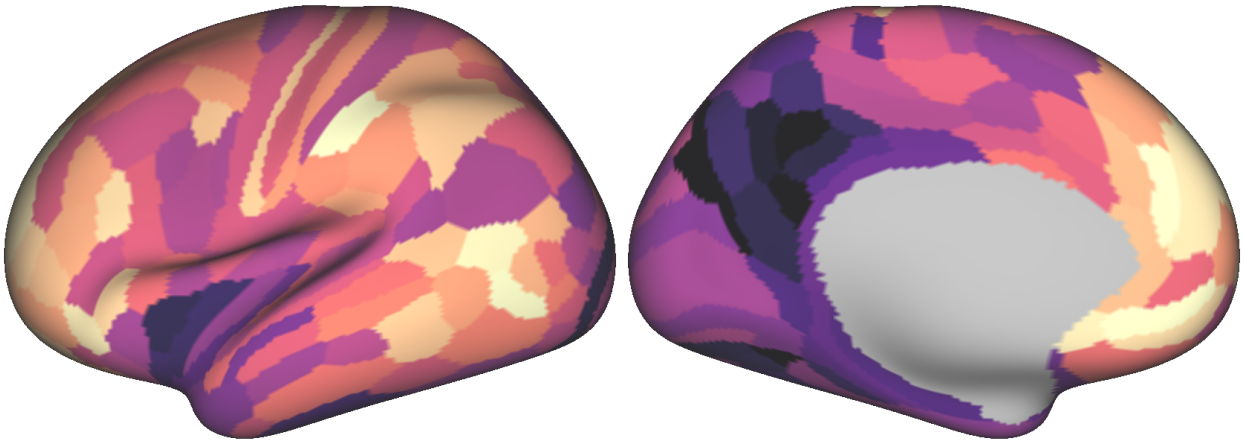


Fig. 1.4: One randomly generated surrogate map.

```
base = Base(x=brain_map_file, D=dist_mat_file, resample=True)
```

The surrogate map illustrated above, had it been generated using `resample=True`, is shown below for comparison:

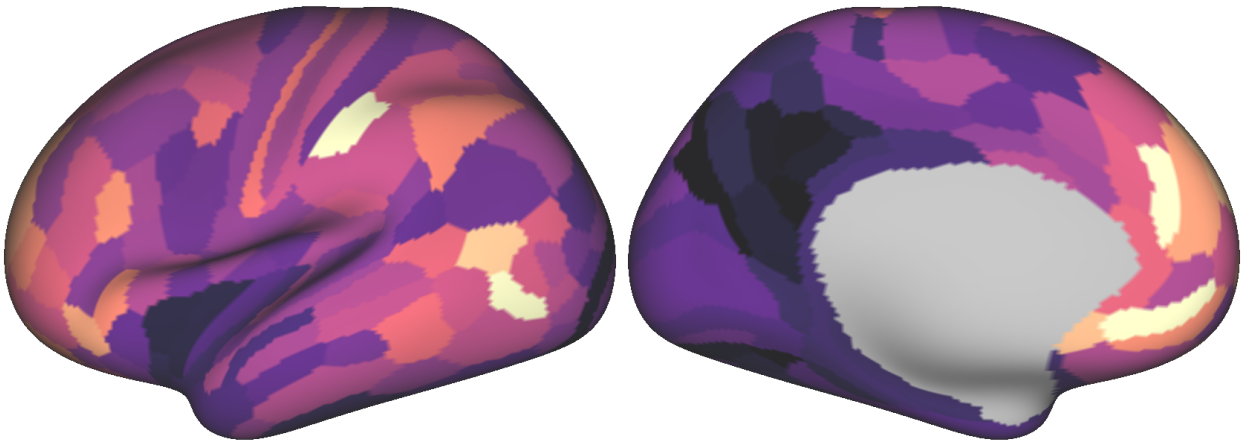


Fig. 1.5: The surrogate brain map above, with values resampled from the empirical map.

Note that using `resample=True` will in general reduce the degree to which the surrogate maps' autocorrelation matches the autocorrelation in the empirical map. However, this discrepancy tends to be small for parcellated brain maps, and tends to be larger for brain maps whose values are more strongly non-normal.

Note: Shameless plug: the plots above were auto-generated using our `wbplot` package, available through both pip and GitHub. `wbplot` currently only supports cortical data, and parcellated data must be in the HCP's MMP parcellation.

Keyword arguments to `brainsmash.mapgen.base.Base`

`deltas` np.ndarray or list[float], default [0.1,0.2,..,0.9] The proportion of neighbors to include during the smoothing step, in the interval (0, 1]. This parameter specifies the different smoothing neighborhood sizes which are iterated over during the variogram optimization.

`kernel` str, default 'exp' The functional form of the smoothing kernel:

- 'gaussian' : Gaussian function
- 'exp' : Exponential decay function
- 'invdist' : Inverse distance
- 'uniform' : Uniform weights (distance independent)

`pv` int, default 25 Percentile of the pairwise distance distribution at which to truncate during variogram fitting. The inclusion of this parameter is motivated by the fact that at large distances, pairwise variability is primarily driven by noise.

`nh` int, default 25 The number of uniformly spaced distance intervals within which to compute variance when constructing variograms. This parameter governs the granularity of your variogram. For noisy brain maps, this parameter should be small enough such that the variogram is smooth and continuous.

`resample` bool, default False Resample surrogate maps' values from empirical brain map, to preserve the distribution of values in each surrogate map. This may produce surrogate maps with poorer fits to the empirical map's variogram.

b float or None, default None The bandwidth of the Gaussian kernel used to smooth the variogram. The variogram isn't particularly sensitive to this parameter, but it's included anyways. If this parameter is None, by default the bandwidth is set to three times the variogram distance interval (see `nh` above).

1.3.3 Dense surrogate maps

Next, we'll demonstrate how to use BrainSMASH to generate surrogate maps for dense (i.e., vertex- or voxel-wise) empirical brain maps, which is a little more tricky. Dense-level data are problematic because of their memory burden — a pairwise distance matrix for data in standard 32k resolution requires more than 4GB of memory if read in all at once from file.

To circumvent these memory issues, we've developed a second core implementation which utilizes memory-mapped arrays and random sampling to avoid loading all of the data into memory at once. However, users with sufficient memory resources and/or supercomputer access are encouraged to use the `Base` implementation described above.

Again, we'll assume that the user already has a brain map and distance matrix saved locally as text files (or downloaded from here). **Update as of Nov. 24, 2020**: Per many requests from the community, BrainSMASH now includes functionality for users working with volumetric data at the whole-brain level. For details, please see the *whole-brain volumetric example*.

Creating memory-mapped arrays

Prior to simulating surrogate maps, you'll need to convert the distance matrix to a memory-mapped binary file, which can be easily achieved in the following way:

```
from brainsmash.mapgen.memmap import txt2memmap
dist_mat_fin = "LeftDenseGeodesicDistmat.txt"  # input text file
output_dir = "."  # directory to which output binaries are written
output_files = txt2memmap(dist_mat_fin, output_dir, maskfile=None, delimiter=' ')
```

The latter two keyword arguments are shown using their default values. If your text files are comma-delimited, for example, use `delimiter=','` instead. Moreover, if you wish to use only a subset of all brain regions, you may also specify a mask (as a path to a neuroimaging file) using the `maskfile` argument.

The return value `output_files` in the code block above is a `dict` type object that will look something like:

```
output_files = {'distmat': '/pathto/output_dir/distmat.npy',
                'index': '/pathto/output_dir/index.npy'}
```

These two files are required inputs to the `brainsmash.mapgen.sampled.Sampled` class.

Note: For additional computational speed-up, `distmat.npy` is sorted by `brainsmash.mapgen.memmap.txt2memmap` before it is written to file; the second file, `index.npy`, is required because it contains the indices which were used to sort the distance matrix.

This text to memory-mapped array conversion only ever needs to be run once for a given distance matrix.

Finally, to generate surrogate maps, we import the `brainsmash.mapgen.sampled.Sampled` class and create an instance by passing our brain map, memory-mapped distance matrix, and memory-mapped index file as arguments:

```
from brainsmash.mapgen.sampled import Sampled
brain_map_file = "LeftDenseMyelin.txt"  # use absolute paths if necessary!
dist_mat_mmap = output_files['distmat']
index_mmap = output_files['index']
sampled = Sampled(brain_map_file, dist_mat_mmap, index_mmap)
```

We then randomly generate surrogate maps with a call to the class instance:

```
surrogates = sampled(n=10)
```

Here, as above, we've implicitly left all keyword arguments – one of which is `resample` – left as their default values. The three images analogous to those shown above, illustrating the dense maps on the cortical surface, are shown below:

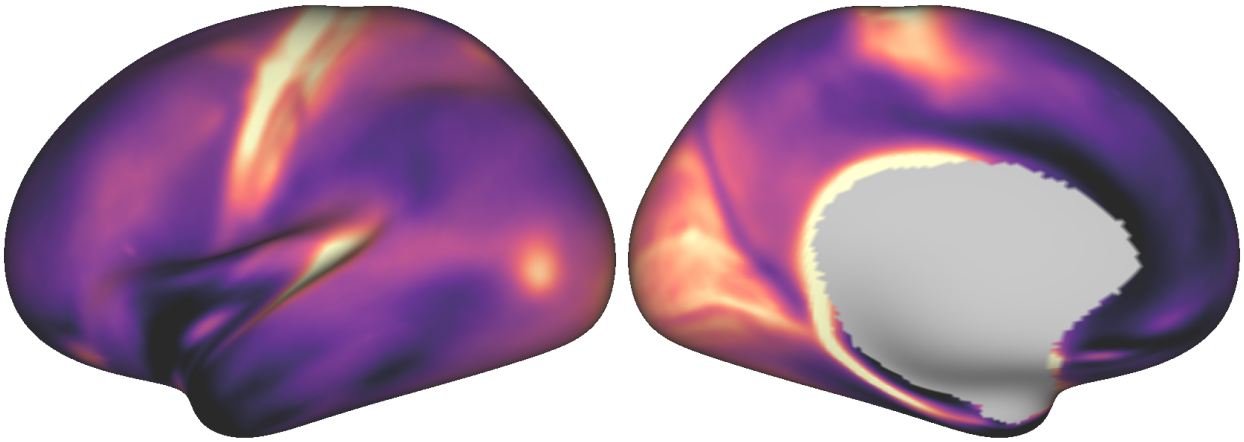


Fig. 1.6: The dense empirical T1w/T2w map.

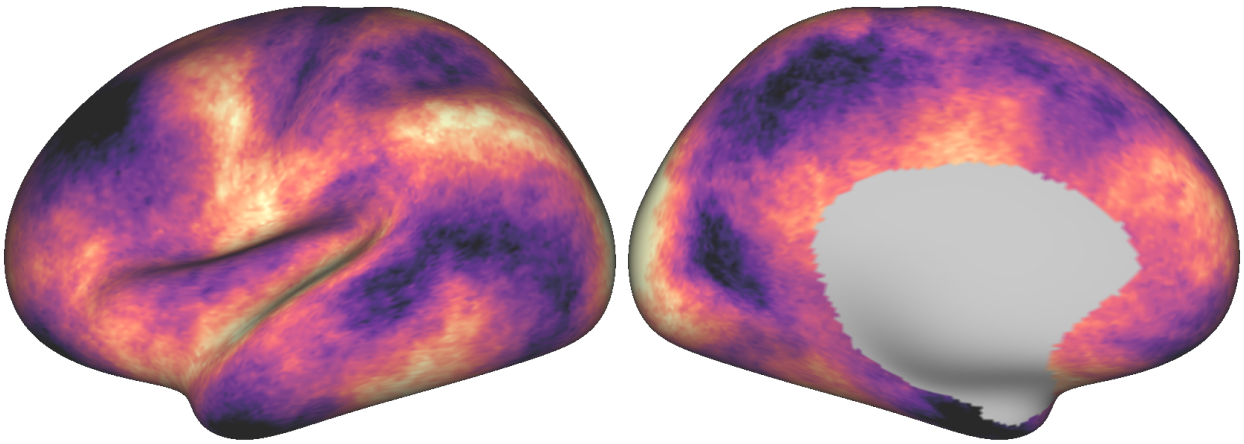


Fig. 1.7: One randomly generated dense surrogate brain map.

Keyword arguments to `brainsmash.mapgen.sampled.Sampled`

`ns` int, default 500 The number of randomly sampled brain areas used to generate a surrogate map.

`knn` int, default 1000 Let **D** be the pairwise distance matrix. Assume each row of **D** has been sorted, in ascending order. Then, because spatial autocorrelation is primarily a local effect, use only **D[:,:knn]**.

`deltas` np.ndarray or list[float], default [0.3,0.5,0.7,0.9] See *above*. Note that fewer values are iterated over by default than in the `Base` class. Users with more time and/or patience are encouraged to expand the default list, as it may improve your surrogate maps.

`kernel` str, default 'exp' See *above*.

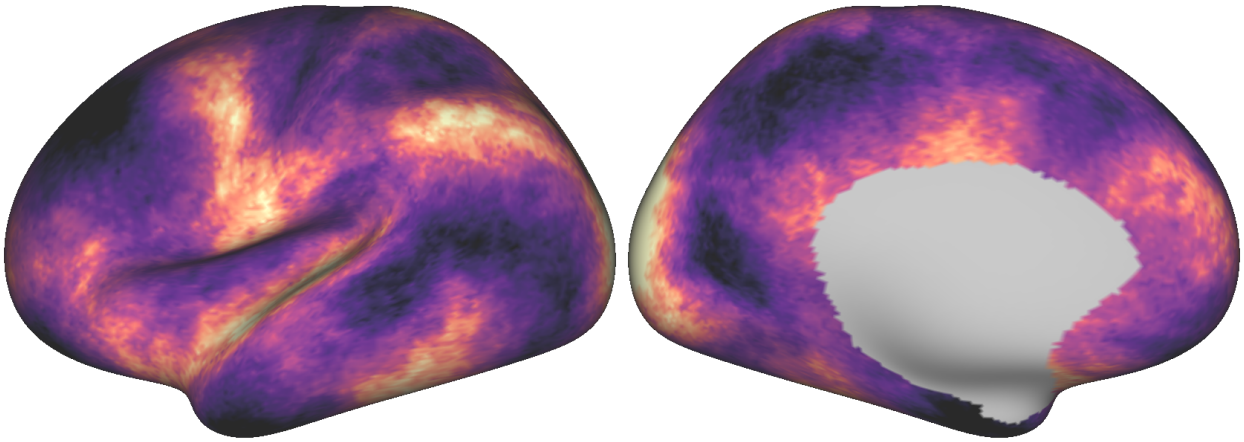


Fig. 1.8: The dense surrogate brain map above, with values resampled from the empirical map.

pv int, default 70 See above. Note that this parameter is by default larger than for the `Base` class; this is in part because of the `knn` parameter above (which is used internally to reduce the distance matrix prior to determining `pv`.

nh int, default 25 See above.

resample bool, default False See *above*.

b float or None, default None See *above*.

Note: Dense data may be used with `brainsmash.mapgen.base.Base` – the examples are primarily partitioned in this way for illustration (but also in anticipation of users' local memory constraints).

In general, the `Sampled` class has much more parameter sensitivity. You may need to adjust these parameters to get reliable variogram fits. However, you may use the functions in the *variogram evaluation* module, which we turn to next, to validate your variogram fits.

1.3.4 Evaluating variogram fits

To assess the reliability of your surrogate maps, BrainSMASH includes functionality to compare surrogate maps' variograms to the target brain map's variogram:

```
from brainsmash.mapgen.eval import base_fit
# from brainsmash.utils.eval import sampled_fit  analogous function for Sampled class
base_fit(brain_map_file, dist_mat_file, nsurr=100)
```

For well-chosen parameters, the code above will produce a plot that looks something like:

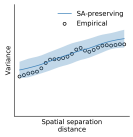


Fig. 1.9: Assessing the surrogate maps' fit to the empirical data.

Shown above is the mean and standard deviation across 100 surrogates. Optional keyword arguments to the base and sampled class can be passed in the calls to the respective evaluation functions – for example, if you want to assess how changing model parameters influences your surrogates maps' variogram fits.

Note: When using `brainsmash.mapgen.eval.sampled_fit`, you must specify the memory-mapped `index` file in addition to the brain map and distance matrix files (see *above*).

1.3.5 Workbench users

The functionality described below is intended for users using GIFTI- and CIFTI-format surface-based neuroimaging files.

Neuroimaging data I/O

To load data from a neuroimaging file into Python, you may use `brainsmash.utils.dataio.load`. For example:

```
from brainsmash.utils.dataio import load
objective = "/path/to/myimage.dscalar.nii"
x = load(objective)  # type(x) == numpy.ndarray
```

Computing a cortical distance matrix

To construct a geodesic distance matrix for a cortical hemisphere, you can do the following:

```
from brainsmash.workbench.geo import cortex
surface = "/path/to/S1200.L.midthickness_MSMAll.32k_fs_LR.surf.gii"
cortex(surface=surface, outfile="/pathto/LeftDenseGeodesicDistmat.txt", euclid=False)
```

Note that this function takes approximately two hours to run for standard 32k surface meshes. To compute 3D Euclidean distances instead of surface-based geodesic distances, simply pass `euclid=True`.

To compute a parcellated geodesic distance matrix, you could then do:

```
from brainsmash.workbench.geo import parcellate
infile = "/path/to/LeftDenseGeodesicDistmat.txt"
outfile = "/path/to/LeftParcelGeodesicDistmat.txt"
dlabel = "Q1-Q6_RelatedValidation210.CorticalAreas_dil_Final_Final_Areas_Group_Colors.
↪32k_fs_L.dlabel.nii"
parcellate(infile, dlabel, outfile)
```

This code takes half an hour or less to run for the HCP MMP1.0. Note that the number of elements in `dlabel` must equal the number of rows/columns of your distance matrix. If you had a whole-brain parcellation file and needed to isolate the left cortical hemisphere, for example, you could do:

```
wb_command -cifti-separate yourparcellation_LR.dlabel.nii COLUMN -label CORTEX_LEFT␣
↪yourparcellation_L.label.gii
```

You will then need to convert this GIFTI file to a CIFTI file:

```
wb_command -cifti-create-label yourparcellation_L.dlabel.nii -left-label␣
↪yourparcellation_L.label.gii
```

For more information, see the -cifti-separate and -cifti-create-label documentation.

Computing a subcortical distance matrix

To compute a Euclidean distance matrix for subcortex, you could do the following:

```
from brainsmash.workbench.geo import subcortex
image_file = "/path/to/image_with_subcortical_volumes.dscalar.nii"
subcortex(outfile="/path/to/SubcortexDenseEuclidDistmat.txt", image_file=image_file)
```

Only three-dimensional Euclidean distance is currently implemented for subcortex. If you wish to create surrogate maps for a single subcortical structure, you can either generate your own mask file and pass it to `brainsmash.mapgen.memmap.txt2memmap`, or follow the procedure described *here*.

Note: If you mask your distance matrix, don't forget to mask your brain map as well. One way this can be achieved is using `brainsmash.workbench.io.image2txt`.

1.4 Examples

1.4.1 Cortical Hemisphere

Here, we'll perform an analysis similar to those included in our pre-print, using our example data.

First, create an instance of the `Base` class and generate 1000 surrogate maps:

```
from brainsmash.mapgen.base import Base
import numpy as np

# load parcellated structural neuroimaging maps
myelin = np.loadtxt("LeftParcelMyelin.txt")
thickness = np.loadtxt("LeftParcelThickness.txt")

# instantiate class and generate 1000 surrogates
gen = Base(myelin, "LeftParcelGeodesicDistmat.txt")  # note: can pass numpy arrays as␣
↪well as filenames
surrogate_maps = gen(n=1000)
```

Next, compute the Pearson correlation between each surrogate myelin map and the empirical cortical thickness map:

```
from brainsmash.mapgen.stats import pearsonr, pairwise_r

surrogate_brainmap_corrs = pearsonr(thickness, surrogate_maps).flatten()
surrogate_pairwise_corrs = pairwise_r(surrogate_maps, flatten=True)
```

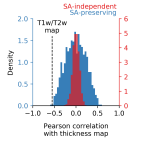
Repeat using randomly shuffled surrogate myelin maps:

```
naive_surrogates = np.array([np.random.permutation(myelin) for _ in range(1000)])
naive_brainmap_corrs = pearsonr(thickness, naive_surrogates).flatten()
naive_pairwise_corrs = pairwise_r(naive_surrogates, flatten=True)
```

Now plot the results:

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

sac = '#377eb8'  # autocorr-preserving
rc = '#e41a1c'  # randomly shuffled
bins = np.linspace(-1, 1, 51)  # correlation b

# this is the empirical statistic we're creating a null distribution for
test_stat = stats.pearsonr(myelin, thickness)[0]

fig = plt.figure(figsize=(3, 3))
ax = fig.add_axes([0.2, 0.25, 0.6, 0.6])  # autocorr preserving
ax2 = ax.twinx()  # randomly shuffled

# plot the data
ax.axvline(test_stat, 0, 0.8, color='k', linestyle='dashed', lw=1)
ax.hist(surrogate_brainmap_corrs, bins=bins, color=sac, alpha=1,
    density=True, clip_on=False, zorder=1)
ax2.hist(naive_brainmap_corrs, bins=bins, color=rc, alpha=0.7,
    density=True, clip_on=False, zorder=2)

# make the plot nice...
ax.set_xticks(np.arange(-1, 1.1, 0.5))
ax.spines['left'].set_color(sac)
ax.tick_params(axis='y', colors=sac)
ax2.spines['right'].set_color(rc)
ax2.tick_params(axis='y', colors=rc)
ax.set_ylim(0, 2)
ax2.set_ylim(0, 6)
ax.set_xlim(-1, 1)
[s.set_visible(False) for s in [
    ax.spines['top'], ax.spines['right'], ax2.spines['top'], ax2.spines['left']]]
ax.text(0.97, 1.1, 'SA-independent', ha='right',va='bottom',
    color=rc, transform=ax.transAxes)
ax.text(0.97, 1.03, 'SA-preserving', ha='right', va='bottom',
    color=sac, transform=ax.transAxes)
ax.text(test_stat, 1.65, "T1w/T2w\nmap", ha='center', va='bottom')
ax.text(0.5, -0.2, "Pearson correlation\nwith thickness map",
    ha='center', va='top', transform=ax.transAxes)
ax.text(-0.3, 0.5, "Density", rotation=90, ha='left', va='center', transform=ax.
↪transAxes)
plt.show()
```

Executing the above code produces the following figure:



We can plot a couple surrogate maps on the cortical surface using wbplot:

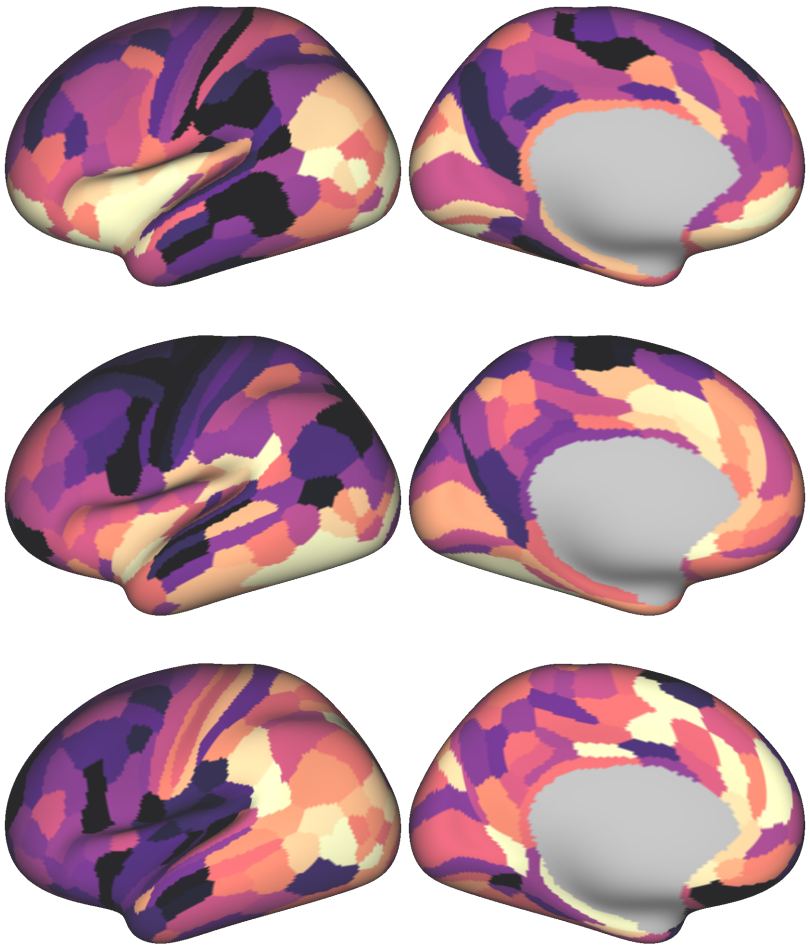
```
from wbplot import pscalar

def vrange(x):
    return (np.percentile(x, 5), np.percentile(x, 95))
```

(continues on next page)

(continued from previous page)

```
for i in range(3):
    y = surrogate_maps[i]
    pscalar(
        file_out="surrogate_{}".format(i+1),
        pscalars=y,
        orientation='landscape',
        hemisphere='left',
        vrange=vrange(y),
        cmap='magma')
```

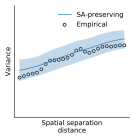
Executing the above code produces the following three images:



We'll assess our surrogate maps' reliability using their fit to the parcellated myelin map's variogram:

```
from brainsmash.mapgen.eval import base_fit

base_fit(
    x="LeftParcelMyelin.txt",
    D="LeftParcelGeodesicDistmat.txt",
    nsurr=1000,
    nh=25,  # these are default kwargs, but shown here for demonstration
    deltas=np.arange(0.1, 1, 0.1),
    pv=25)  # kwargs are passed to brainsmash.mapgen.base.Base
```

Executing the code above produces the following plot:



The surrogate maps exhibit the same autocorrelation structure as the empirical brain map.

Finally, we'll compute non-parametric *p*-values using our two different null distributions:

```
from brainsmash.mapgen.stats import nonparp

print("Spatially naive p-value:", nonparp(test_stat, naive_brainmap_corrs))
print("SA-corrected p-value:", nonparp(test_stat, surrogate_brainmap_corrs))
```

The two *p*-values for this example come out to `P < 0.001` and `P=0.001`, respectively.

1.4.2 Unilateral Subcortical Structure

For a subcortical analysis you'll typically need:

- A subcortical distance matrix
- A subcortical brain map
- A mask corresponding to a structure of interest

We'll assume you use Connectome Workbench-style files and that you want to isolate one particular anatomical structure.

Note: If you already have a distance matrix and a brain map for your subcortical structure of interest, the workflow is identical to the cortical examples in *Getting Started*.

If you haven't already computed a subcortical distance matrix or downloaded our pre-computed version, please follow *these steps*.

To isolate one subcortical structure from a whole-brain `dscalar` file, first do:

```
wb_command -cifti-export-dense-mapping yourfile.dscalar.nii COLUMN -volume-all output.
↪txt -structure
```

We can then use the information contained in this file to isolate one particular structure, e.g. the left cerebellum:

```
from brainsmash.utils.dataio import load
import numpy as np
import pandas as pd

# Input files
image = "/path/to/yourfile.dscalar.nii"
wb_output = "output.txt"

# Load the output of the above command
df = pd.read_table(wb_output, header=None, index_col=0, sep=' ',
         names=['structure', 'mni_i', 'mni_j', 'mni_k']).rename_axis('index')
```

(continues on next page)

(continued from previous page)

```
# Get indices for left cerebellum
indices = df[df['structure'] == 'CEREBELLUM_LEFT'].index.values

# Create a binary mask
mask = np.ones(31870)  # volume has 31870 CIFTI indices in standard 91k mesh
indices -= 59412  # first 59412 indices in whole-brain maps are cortical
mask[indices] = 0
np.savetxt("mask.txt", mask)  # this mask has right dimension for distmat

# Also saved a masked copy of the image
image_data = load(image)
indices += 59412  # assuming image data is whole-brain!
masked_image = image_data[indices]
np.savetxt("masked_image.txt", masked_image)
```

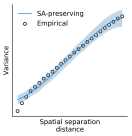
Next, we'll need to sort and memory-map our distance matrix, but only for the pairwise distances between left cerebellar voxels:

```
from brainsmash.mapgen.memmap import txt2memmap

# Input files
image = "masked_image.txt"
mask = "mask.txt"
distmat = "/path/to/SubcortexDenseEuclideanDistmat.txt"

output_files = txt2memmap(distmat, output_dir=".", maskfile=mask, delimiter=' ')
```

Now, we can use the output files to instantiate our surrogate map generator. Here, we'll also use the keyword arguments which were used in our study for left cerebellum. First, we'll validate the variogram fit using these parameters:

```
from brainsmash.mapgen.eval import sampled_fit

x = "masked_image.txt"
distmat = output_files['distmat']
index = output_files['index']

kwargs = {'ns': 500,
          'knn': 1500,
          'nh': 25,
          'deltas': [0.3, 0.5, 0.7, 0.9],
          'pv': 70
          }

sampled_fit(x, distmat, index, nsurr=20, **kwargs)
```

This produces the following plot:



Having confirmed that the fit looks good, we simulate cerebellar surrogate maps with a call to the surrogate map generator:

```
from brainsmash.mapgen.sampled import Sampled

gen = Sampled(x, distmat, index, **kwargs)
surrogate_maps = gen(n=100)
```

1.4.3 Whole-Brain Volume

For this analysis you will need:

- A text file containing voxel coordinates (with shape N rows by 3 columns)
- A text file containing N brain map values

Your favorite neuroimaging software should be able to generate these files (e.g., AFNI).

Note: Do not include every voxel in the volume, but rather the voxels corresponding to your region of interest (e.g., the brain).

First, you will need to generate memory-mapped distance matrix files. This is achieved using `brainsmash.workbench.geo.volume`, as in the code below:

```
from brainsmash.workbench.geo import volume

coord_file = "/path/to/voxel_coordinates.txt"
output_dir = "/some/directory/"

filenames = volume(coord_file, output_dir)
```

A dictionary containing the names of the newly generated files is returned by the function, which are used in the example below. Prior to generating surrogate maps, we first visually inspect the variogram fit:

```
from brainsmash.mapgen.eval import sampled_fit

brain_map = "/some_path_to/brain_map.txt"

# These are three of the key parameters affecting the variogram fit
kwargs = {'ns': 500,
          'knn': 1500,
          'pv': 70
          }

# Running this command will generate a matplotlib figure
sampled_fit(brain_map, filenames['D'], filenames['index'], nsurr=10, **kwargs)
```

If you are unhappy with the fit, it is recommended that you try varying the three parameters included in the keyword argument dictionary above (i.e., `ns`, `knn`, and `pv`).

Note: A few users have reported a systematic overestimation of variance in their surrogate maps' variograms. This suggests that further parameter tuning is necessary. However, it may also indicate a violation of the underlying assumption of stationary, which is more likely to be violated at the whole-brain level than within a single contiguous brain structure.

Having selected our parameter values and stored them in the `kwargs` dictionary, we can now randomly generate surrogate brain maps with spatial autocorrelation that is matched to our target brain map:

```
from brainsmash.mapgen.sampled import Sampled

gen = Sampled(x=brain_map, D=filenames['D'], index=filenames['index'], **kwargs)
surrogate_maps = gen(n=1000)
```

These surrogate maps can then be used to test the null hypothesis that any random yet spatially autocorrelated brain map would yield a comparable or more extreme statistical result.

1.5 API Documentation

1.5.1 Generating Surrogate Brain Maps

Base Implementation

`Base`(x, D[, deltas, kernel, pv, nh, ...])	Base implementation of map generator.

Sampled Implementation

`Sampled`(x, D, index[, ns, pv, nh, knn, b, ...])	Sampling implementation of map generator.

Variogram Evaluation

`base_fit`(x, D[, nsurr, return_data])	Evaluate variogram fits for Base class.
`sampled_fit`(x, D, index[, nsurr, return_data])	Evaluate variogram fits for Sampled class.

Smoothing Kernels

`exp`(d)	Exponentially decaying kernel which truncates at e^{-1}.
`gaussian`(d)	Gaussian kernel which truncates at one standard deviation.
`invdist`(d)	Inverse distance kernel.
`uniform`(d)	Uniform (i.e., distance independent) kernel.

Creating Memory-Mapped Arrays

`txt2memmap`(dist_file, output_dir[, ...])	Export distance matrix to memory-mapped array.
`load_memmap`(filename)	Load a memory-mapped array.

Statistical Methods

`pearsonr`(X, Y)	Multi-dimensional Pearson correlation between rows of *X* and *Y*.
`pairwise_r`(X[, flatten])	Compute pairwise Pearson correlations between rows of *X*.
`nonparp`(stat, dist)	Compute two-sided non-parametric p-value.

1.5.2 Connectome Workbench Utilities

Constructing Distance Matrices

`cortex`(surface, outfile[, euclid, dlabel, ...])	Calculates surface distances for *surface* mesh and saves to *outfile*.
`subcortex`(fout[, image_file, dlabel, ...])	Compute inter-voxel Euclidean distance matrix.
`parcellate`(infile, dlabel_file, outfile[, ...])	Parcellate a dense distance matrix.

Neuroimaging Data I/O

`image2txt`(image_file, outfile[, maskfile, ...])	Export neuroimaging data to txt file.

1.6 References

If you use BrainSMASH in your research, please cite the following study:

Burt, J.B., Helmer, M., Shinn, M.W., Anticevic, A., Murray, J.D. Generative modeling of brain maps with spatial autocorrelation. Neuroimage, 220 (2020).

The methodology used to generate surrogate maps in BrainSMASH was adapted from:

Viladomat, J., Mazumder, R., McInturff, A., McCauley, D. J., & Hastie, T. (2014). Assessing the significance of global and local correlations under spatial autocorrelation: A nonparametric approach. Biometrics, 70(2), 409-418.

CHAPTER

TWO

CORE DEVELOPMENT TEAM

- Joshua B. Burt, Yale University
- John D. Murray, Yale University

CHAPTER

THREE

CONTRIBUTORS

- Ross Markello, Montreal Neurological Institute

CHAPTER

FOUR

LICENSE

The BrainSMASH source code is available under the GNU General Public License v3.0.

CHAPTER

FIVE

SUPPORT

If you run into a problem or identify a bug in the source code, please send the authors an email or create a new issue on GitHub.