PHATE Documentation

Release 1.0.4

Daniel Burkhardt, Krishnaswamy Lab, Yale University

Mar 18, 2020

Contents

PHATE (Potential of Heat-diffusion for Affinity-based Trajectory Embedding) is a tool for visualizing high dimensional data. PHATE uses a novel conceptual framework for learning and visualizing the manifold to preserve both local and global distances.

To see how PHATE can be applied to datasets such as facial images and single-cell data from human embryonic stem cells, check out our Nature Biotechnology publication.

Moon, van Dijk, Wang, Gigante et al. **Visualizing Transitions and Structure for Biological Data Exploration**. 2019. *Nature Biotechnology*.

CHAPTER 1

Installation

1.1 Python installation

1.1.1 Installation with *pip*

The Python version of PHATE can be installed using:

```
pip install --user phate
```

1.1.2 Installation from source

The Python version of PHATE can be installed from GitHub by running the following from a terminal:

```
git clone --recursive git://github.com/KrishnaswamyLab/PHATE.git
cd Python
python setup.py install --user
```

1.2 MATLAB installation

1. The MATLAB version of PHATE can be accessed using:

   ```
   git clone git://github.com/KrishnaswamyLab/PHATE.git
   cd PHATE/Matlab
   ```

2. Add the PHATE/Matlab directory to your MATLAB path and run any of our *test* scripts to get a feel for PHATE.

1.3 R installation

In order to use PHATE in R, you must also install the Python package.

1.3.1 Installation from CRAN and PyPi

Install *phateR* from CRAN by running the following code in R:

```
install.packages("phateR")
```

Install *phate* in Python by running the following code from a terminal:

```
pip install --user phate
```

1.3.2 Installation with *devtools* and *reticulate*

The development version of PHATE can be installed directly from R with *devtools*:

```
if (!suppressWarnings(require(devtools))) install.packages("devtools")
devtools::install_github("KrishnaswamyLab/phateR")
```

If you have the development version of *reticulate*, you can also install *phate* in Python by running the following code in R:

```
devtools::install_github("rstudio/reticulate")
reticulate::py_install("phate")
```

1.3.3 Installation from source

The latest source version of PHATE can be accessed by running the following in a terminal:

```
git clone --recursive git://github.com/KrishnaswamyLab/PHATE.git
cd PHATE/phateR
R CMD INSTALL
cd ../Python
python setup.py install --user
```

If the *phateR* folder is empty, you have may forgotten to use the *–recursive* option for *git clone*. You can rectify this by running the following in a terminal:

```
cd PHATE
git submodule init
git submodule update
cd phateR
R CMD INSTALL
cd ../Python
python setup.py install --user
```

CHAPTER 2

Tutorial

To run PHATE on your dataset, create a PHATE operator and run *fit_transform*. Here we show an example with an artificial tree:

```
import phate
import scprep
tree_data, tree_clusters = phate.tree.gen_dla()
phate_operator = phate.PHATE(k=15, t=100)
tree_phate = phate_operator.fit_transform(tree_data)
scprep.plot.scatter2d(tree_phate, c=tree_clusters)
phate_operator.set_params(n_components=3)
tree_phate = phate_operator.transform()
scprep.plot.rotate_scatter3d(tree_phate, c=tree_clusters)
```

A demo on PHATE usage and visualization for single cell RNA-seq data can be found in this Jupyter notebook. A second tutorial is available here which works with the artificial tree shown above in more detail. You can also access interactive versions of these tutorials on Google Colaboratory: single cell RNA seq, artificial tree.

CHAPTER 3

API

3.1 PHATE

Potential of Heat-diffusion for Affinity-based Trajectory Embedding (PHATE)

class phate.phate.**PHATE**(*n_components=2*, *knn=5*, *decay=40*, *n_landmark=2000*, *t='auto'*, *gamma=1*, *n_pca=100*, *mds_solver='sgd'*, *knn_dist='euclidean'*, *mds_dist='euclidean'*, *mds='metric'*, *n_jobs=1*, *random_state=None*, *verbose=1*, *potential_method=None*, *alpha_decay=None*, *njobs=None*, *k=None*, *a=None*, ***kwargs*)

Bases: `sklearn.base.BaseEstimator`

PHATE operator which performs dimensionality reduction.

Potential of Heat-diffusion for Affinity-based Trajectory Embedding (PHATE) embeds high dimensional single-cell data into two or three dimensions for visualization of biological progressions as described in Moon et al, 2017[1].

Parameters

- **n_components** (*int, optional, default: 2*) – number of dimensions in which the data will be embedded
- **knn** (*int, optional, default: 5*) – number of nearest neighbors on which to build kernel
- **decay** (*int, optional, default: 40*) – sets decay rate of kernel tails. If None, alpha decaying kernel is not used
- **n_landmark** (*int, optional, default: 2000*) – number of landmarks to use in fast PHATE
- **t** (*int, optional, default: 'auto'*) – power to which the diffusion operator is powered. This sets the level of diffusion. If 'auto', t is selected according to the knee point in the Von Neumann Entropy of the diffusion operator

[1] Moon KR, van Dijk D, Zheng W, *et al.* (2017), *PHATE: A Dimensionality Reduction Method for Visualizing Trajectory Structures in High-Dimensional Biological Data*, BioRxiv.

- **gamma** (*float, optional, default: 1*) – Informational distance constant between -1 and 1. *gamma=1* gives the PHATE log potential, *gamma=0* gives a square root potential.
- **n_pca** (*int, optional, default: 100*) – Number of principal components to use for calculating neighborhoods. For extremely large datasets, using n_pca < 20 allows neighborhoods to be calculated in roughly log(n_samples) time.
- **mds_solver** (*{'sgd', 'smacof'}, optional (default: 'sgd')*) – which solver to use for metric MDS. SGD is substantially faster, but produces slightly less optimal results. Note that SMACOF was used for all figures in the PHATE paper.
- **knn_dist** (*string, optional, default: 'euclidean'*) – recommended values: 'euclidean', 'cosine', 'precomputed' Any metric from *scipy.spatial.distance* can be used distance metric for building kNN graph. Custom distance functions of form *f(x, y) = d* are also accepted. If 'precomputed', *data* should be an n_samples x n_samples distance or affinity matrix. Distance matrices are assumed to have zeros down the diagonal, while affinity matrices are assumed to have non-zero values down the diagonal. This is detected automatically using *data[0,0]*. You can override this detection with *knn_dist='precomputed_distance'* or *knn_dist='precomputed_affinity'*.
- **mds_dist** (*string, optional, default: 'euclidean'*) – Distance metric for MDS. Recommended values: 'euclidean' and 'cosine' Any metric from *scipy.spatial.distance* can be used. Custom distance functions of form *f(x, y) = d* are also accepted
- **mds** (*string, optional, default: 'metric'*) – choose from ['classic', 'metric', 'nonmetric']. Selects which MDS algorithm is used for dimensionality reduction
- **n_jobs** (*integer, optional, default: 1*) – The number of jobs to use for the computation. If -1 all CPUs are used. If 1 is given, no parallel computing code is used at all, which is useful for debugging. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used. Thus for n_jobs = -2, all CPUs but one are used
- **random_state** (*integer or numpy.RandomState, optional, default: None*) – The generator used to initialize SMACOF (metric, nonmetric) MDS If an integer is given, it fixes the seed Defaults to the global *numpy* random number generator
- **verbose** (*int* or *boolean*, optional (default: 1)) – If *True* or > *0*, print status messages
- **potential_method** (*deprecated.*) – Use *gamma=1* for log transformation and *gamma=0* for square root transformation.
- **alpha_decay** (*deprecated.*) – Use *decay=None* to disable alpha decay
- **njobs** (*deprecated.*) – Use n_jobs to match *sklearn* standards
- **k** (Deprecated for *knn*) –
- **a** (Deprecated for *decay*) –
- **kwargs** (additional arguments for *graphtools.Graph*) –

X

Type array-like, shape=[n_samples, n_dimensions]

embedding

Stores the position of the dataset in the embedding space

Type array-like, shape=[n_samples, n_components]

diff_op
The diffusion operator built from the graph

Type array-like, shape=[n_samples, n_samples] or [n_landmark, n_landmark]

graph
The graph built on the input data

Type graphtools.base.BaseGraph

optimal_t
The automatically selected t, when t = 'auto'. When t is given, optimal_t is None.

Type int

Examples

```
>>> import phate
>>> import matplotlib.pyplot as plt
>>> tree_data, tree_clusters = phate.tree.gen_dla(n_dim=100, n_branch=20,
...                                               branch_length=100)
>>> tree_data.shape
(2000, 100)
>>> phate_operator = phate.PHATE(knn=5, decay=20, t=150)
>>> tree_phate = phate_operator.fit_transform(tree_data)
>>> tree_phate.shape
(2000, 2)
>>> phate.plot.scatter2d(tree_phate, c=tree_clusters)
```

References

diff_op
The diffusion operator calculated from the data

diff_potential
Interpolates the PHATE potential to one entry per cell

This is equivalent to calculating infinite-dimensional PHATE, or running PHATE without the MDS step.

Returns diff_potential

Return type ndarray, shape=[n_samples, min(n_landmark, n_samples)]

fit(*X*)
Computes the diffusion operator

Parameters X (*array, shape=[n_samples, n_features]*) – input data with *n_samples* samples and *n_dimensions* dimensions. Accepted data types: *numpy.ndarray*, *scipy.sparse.spmatrix*, *pd.DataFrame*, *anndata.AnnData*. If *knn_dist* is 'precomputed', *data* should be a n_samples x n_samples distance or affinity matrix

Returns

- **phate_operator** (*PHATE*)
- *The estimator object*

fit_transform(*X*, ***kwargs*)
Computes the diffusion operator and the position of the cells in the embedding space

Parameters

- **X** (*array, shape=[n_samples, n_features]*) – input data with *n_samples* samples and *n_dimensions* dimensions. Accepted data types: *numpy.ndarray*, *scipy.sparse.spmatrix*, *pd.DataFrame*, *anndata.AnnData* If *knn_dist* is 'precomputed', *data* should be a n_samples x n_samples distance or affinity matrix
- **kwargs** (further arguments for *PHATE.transform()*) – Keyword arguments as specified in `transform()`

Returns **embedding** – The cells embedded in a lower dimensional space using PHATE

Return type array, shape=[n_samples, n_dimensions]

get_params(*deep=True*)
Get parameters for this estimator.

Parameters **deep** (*bool, default=True*) – If True, will return the parameters for this estimator and contained subobjects that are estimators.

Returns **params** – Parameter names mapped to their values.

Return type mapping of string to any

reset_mds(***kwargs*)
Deprecated. Reset parameters related to multidimensional scaling

Parameters

- **n_components** (*int, optional, default: None*) – If given, sets number of dimensions in which the data will be embedded
- **mds** (*string, optional, default: None*) – choose from ['classic', 'metric', 'nonmetric'] If given, sets which MDS algorithm is used for dimensionality reduction
- **mds_dist** (*string, optional, default: None*) – recommended values: 'euclidean' and 'cosine' Any metric from scipy.spatial.distance can be used If given, sets the distance metric for MDS

reset_potential(***kwargs*)
Deprecated. Reset parameters related to the diffusion potential

Parameters

- **t** (*int or 'auto', optional, default: None*) – Power to which the diffusion operator is powered If given, sets the level of diffusion
- **potential_method** (*string, optional, default: None*) – choose from ['log', 'sqrt'] If given, sets which transformation of the diffusional operator is used to compute the diffusion potential

set_params(***params*)
Set the parameters on this estimator.

Any parameters not given as named arguments will be left at their current value.

Parameters

- **n_components** (*int, optional, default: 2*) – number of dimensions in which the data will be embedded
- **knn** (*int, optional, default: 5*) – number of nearest neighbors on which to build kernel
- **decay** (*int, optional, default: 40*) – sets decay rate of kernel tails. If None, alpha decaying kernel is not used

- **n_landmark** (*int, optional, default: 2000*) – number of landmarks to use in fast PHATE
- **t** (*int, optional, default: 'auto'*) – power to which the diffusion operator is powered. This sets the level of diffusion. If 'auto', t is selected according to the knee point in the Von Neumann Entropy of the diffusion operator
- **gamma** (*float, optional, default: 1*) – Informational distance constant between -1 and 1. *gamma=1* gives the PHATE log potential, *gamma=0* gives a square root potential.
- **n_pca** (*int, optional, default: 100*) – Number of principal components to use for calculating neighborhoods. For extremely large datasets, using n_pca < 20 allows neighborhoods to be calculated in roughly log(n_samples) time.
- **mds_solver** (*{'sgd', 'smacof'}, optional (default: 'sgd')*) – which solver to use for metric MDS. SGD is substantially faster, but produces slightly less optimal results. Note that SMACOF was used for all figures in the PHATE paper.
- **knn_dist** (*string, optional, default: 'euclidean'*) – recommended values: 'euclidean', 'cosine', 'precomputed' Any metric from *scipy.spatial.distance* can be used distance metric for building kNN graph. Custom distance functions of form *f(x, y) = d* are also accepted. If 'precomputed', *data* should be an n_samples x n_samples distance or affinity matrix. Distance matrices are assumed to have zeros down the diagonal, while affinity matrices are assumed to have non-zero values down the diagonal. This is detected automatically using *data[0,0]*. You can override this detection with *knn_dist='precomputed_distance'* or *knn_dist='precomputed_affinity'*.
- **mds_dist** (*string, optional, default: 'euclidean'*) – recommended values: 'euclidean' and 'cosine' Any metric from *scipy.spatial.distance* can be used distance metric for MDS
- **mds** (*string, optional, default: 'metric'*) – choose from ['classic', 'metric', 'nonmetric']. Selects which MDS algorithm is used for dimensionality reduction
- **n_jobs** (*integer, optional, default: 1*) – The number of jobs to use for the computation. If -1 all CPUs are used. If 1 is given, no parallel computing code is used at all, which is useful for debugging. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used. Thus for n_jobs = -2, all CPUs but one are used
- **random_state** (*integer or numpy.RandomState, optional, default: None*) – The generator used to initialize SMACOF (metric, nonmetric) MDS If an integer is given, it fixes the seed Defaults to the global *numpy* random number generator
- **verbose** (*int* or *boolean*, optional (default: 1)) – If *True* or > *0*, print status messages
- **k** (Deprecated for *knn*) –
- **a** (Deprecated for *decay*) –

Examples

```
>>> import phate
>>> import matplotlib.pyplot as plt
>>> tree_data, tree_clusters = phate.tree.gen_dla(n_dim=50, n_branch=5,
...                                               branch_length=50)
```

(continues on next page)

(continued from previous page)

```
>>> tree_data.shape
(250, 50)
>>> phate_operator = phate.PHATE(k=5, a=20, t=150)
>>> tree_phate = phate_operator.fit_transform(tree_data)
>>> tree_phate.shape
(250, 2)
>>> phate_operator.set_params(n_components=10)
PHATE(a=20, alpha_decay=None, k=5, knn_dist='euclidean', mds='metric',
   mds_dist='euclidean', n_components=10, n_jobs=1, n_landmark=2000,
   n_pca=100, njobs=None, potential_method='log', random_state=None, t=150,
   verbose=1)
>>> tree_phate = phate_operator.transform()
>>> tree_phate.shape
(250, 10)
>>> # plt.scatter(tree_phate[:,0], tree_phate[:,1], c=tree_clusters)
>>> # plt.show()
```

Returns

Return type self

transform(*X=None*, *t_max=100*, *plot_optimal_t=False*, *ax=None*)

Computes the position of the cells in the embedding space

Parameters

- **X** (*array, optional, shape=[n_samples, n_features]*) – input data with *n_samples* samples and *n_dimensions* dimensions. Not required, since PHATE does not currently embed cells not given in the input matrix to *PHATE.fit()*. Accepted data types: *numpy.ndarray*, *scipy.sparse.spmatrix*, *pd.DataFrame*, *anndata.AnnData*. If *knn_dist* is 'precomputed', *data* should be a n_samples x n_samples distance or affinity matrix
- **t_max** (*int, optional, default: 100*) – maximum t to test if *t* is set to 'auto'
- **plot_optimal_t** (*boolean, optional, default: False*) – If true and *t* is set to 'auto', plot the Von Neumann entropy used to select t
- **ax** (*matplotlib.axes.Axes, optional*) – If given and *plot_optimal_t* is true, plot will be drawn on the given axis.

Returns

- **embedding** (*array, shape=[n_samples, n_dimensions]*)
- *The cells embedded in a lower dimensional space using PHATE*

3.2 Clustering

phate.cluster.**kmeans**(*phate_op*, *n_clusters='auto'*, *max_clusters=10*, *random_state=None*, *k=None*, ***kwargs*)

KMeans on the PHATE potential

Clustering on the PHATE operator as introduced in Moon et al. This is similar to spectral clustering.

Parameters

- **phate_op** (phate.PHATE) – Fitted PHATE operator

- **n_clusters** (*int, optional (default: 'auto')*) – Number of clusters. If 'auto', uses the Silhouette score to determine the optimal number of clusters
- **max_clusters** (*int, optional (default: 10)*) – Maximum number of clusters to test if using the Silhouette score.
- **random_state** (*int or None, optional (default: None)*) – Random seed for k-means
- **k** (deprecated for *n_clusters*) –
- **kwargs** (additional arguments for *sklearn.cluster.KMeans*) –

Returns **clusters** – Integer array of cluster assignments

Return type np.ndarray

phate.cluster.**silhouette_score**(*phate_op*, *n_clusters*, *random_state=None*, ***kwargs*)

Compute the Silhouette score on KMeans on the PHATE potential

Parameters

- **phate_op** (phate.PHATE) – Fitted PHATE operator
- **n_clusters** (*int*) – Number of clusters.
- **random_state** (*int or None, optional (default: None)*) – Random seed for k-means

Returns **score**

Return type float

3.3 Plotting

phate.plot.**rotate_scatter3d**(*data*, *filename=None*, *elev=30*, *rotation_speed=30*, *fps=10*, *ax=None*, *figsize=None*, *dpi=None*, *ipython_html='jshtml'*, ***kwargs*)

Create a rotating 3D scatter plot

Builds upon *matplotlib.pyplot.scatter* with nice defaults and handles categorical colors / legends better.

Parameters

- **data** (array-like, *phate.PHATE* or *scanpy.AnnData*) – Input data. Only the first three dimensions are used.
- **filename** (*str, optional (default: None)*) – If not None, saves a .gif or .mp4 with the output
- **elev** (*float, optional (default: 30)*) – Elevation of viewpoint from horizontal, in degrees
- **rotation_speed** (*float, optional (default: 30)*) – Speed of axis rotation, in degrees per second
- **fps** (*int, optional (default: 10)*) – Frames per second. Increase this for a smoother animation
- **ax** (*matplotlib.Axes* or None, optional (default: None)) – axis on which to plot. If None, an axis is created

- **figsize** (*tuple, optional (default: None)*) – Tuple of floats for creation of new *matplotlib* figure. Only used if *ax* is None.
- **dpi** (*number, optional (default: None)*) – Controls the dots per inch for the movie frames. This combined with the figure's size in inches controls the size of the movie. If None, defaults to rcParams["savefig.dpi"]
- **ipython_html** (*{'html5', 'jshtml'}*) – which html writer to use if using a Jupyter Notebook
- ****kwargs** (*keyword arguments*) – See :~func:*phate.plot.scatter3d*.

Returns **ani** – animation object

Return type *matplotlib.animation.FuncAnimation*

Examples

```
>>> import phate
>>> import matplotlib.pyplot as plt
>>> tree_data, tree_clusters = phate.tree.gen_dla(n_dim=100, n_branch=20,
...                                               branch_length=100)
>>> tree_data.shape
(2000, 100)
>>> phate_operator = phate.PHATE(n_components=3, k=5, a=20, t=150)
>>> tree_phate = phate_operator.fit_transform(tree_data)
>>> tree_phate.shape
(2000, 2)
>>> phate.plot.rotate_scatter3d(tree_phate, c=tree_clusters)
```

phate.plot.**scatter**(*x, y, z=None, c=None, cmap=None, s=None, discrete=None, ax=None, legend=None, figsize=None, xticks=False, yticks=False, zticks=False, xticklabels=True, yticklabels=True, zticklabels=True, label_prefix='PHATE', xlabel=None, ylabel=None, zlabel=None, title=None, legend_title='', legend_loc='best', filename=None, dpi=None, **plot_kwargs*)

Create a scatter plot

Builds upon *matplotlib.pyplot.scatter* with nice defaults and handles categorical colors / legends better. For easy access, use *scatter2d* or *scatter3d*.

Parameters

- **x** (*list-like*) – data for x axis
- **y** (*list-like*) – data for y axis
- **z** (*list-like, optional (default: None)*) – data for z axis
- **c** (*list-like or None, optional (default: None)*) – Color vector. Can be a single color value (RGB, RGBA, or named matplotlib colors), an array of these of length n_samples, or a list of discrete or continuous values of any data type. If *c* is not a single or list of matplotlib colors, the values in *c* will be used to populate the legend / colorbar with colors from *cmap*
- **cmap** (*matplotlib* colormap, str, dict or None, optional (default: None)) – matplotlib colormap. If None, uses *tab20* for discrete data and *inferno* for continuous data. If a dictionary, expects one key for every unique value in *c*, where values are valid matplotlib colors (hsv, rbg, rgba, or named colors)
- **s** (*float, optional (default: 1)*) – Point size.

- **discrete** (*bool or None, optional (default: None)*) – If True, the legend is categorical. If False, the legend is a colorbar. If None, discreteness is detected automatically. Data containing non-numeric *c* is always discrete, and numeric data with 20 or less unique values is discrete.
- **ax** (*matplotlib.Axes* or None, optional (default: None)) – axis on which to plot. If None, an axis is created
- **legend** (*bool, optional (default: True)*) – States whether or not to create a legend. If data is continuous, the legend is a colorbar.
- **figsize** (*tuple, optional (default: None)*) – Tuple of floats for creation of new *matplotlib* figure. Only used if *ax* is None.
- **xticks** (*True, False, or list-like (default: False)*) – If True, keeps default x ticks. If False, removes x ticks. If a list, sets custom x ticks
- **yticks** (*True, False, or list-like (default: False)*) – If True, keeps default y ticks. If False, removes y ticks. If a list, sets custom y ticks
- **zticks** (*True, False, or list-like (default: False)*) – If True, keeps default z ticks. If False, removes z ticks. If a list, sets custom z ticks. Only used for 3D plots.
- **xticklabels** (*True, False, or list-like (default: True)*) – If True, keeps default x tick labels. If False, removes x tick labels. If a list, sets custom x tick labels
- **yticklabels** (*True, False, or list-like (default: True)*) – If True, keeps default y tick labels. If False, removes y tick labels. If a list, sets custom y tick labels
- **zticklabels** (*True, False, or list-like (default: True)*) – If True, keeps default z tick labels. If False, removes z tick labels. If a list, sets custom z tick labels. Only used for 3D plots.
- **label_prefix** (*str or None (default: "PHATE")*) – Prefix for all axis labels. Axes will be labelled *label_prefix'1, 'label_prefix'2, etc. Can be overriden by setting 'xlabel*, *ylabel*, and *zlabel*.
- **xlabel** (*str or None (default : None)*) – Label for the x axis. Overrides the automatic label given by label_prefix. If None and label_prefix is None, no label is set.
- **ylabel** (*str or None (default : None)*) – Label for the y axis. Overrides the automatic label given by label_prefix. If None and label_prefix is None, no label is set.
- **zlabel** (*str or None (default : None)*) – Label for the z axis. Overrides the automatic label given by label_prefix. If None and label_prefix is None, no label is set. Only used for 3D plots.
- **title** (*str or None (default: None)*) – axis title. If None, no title is set.
- **legend_title** (*str (default: "")*) – title for the colorbar of legend. Only used for discrete data.
- **legend_loc** (*int or string or pair of floats, default: 'best'*) – Matplotlib legend location. Only used for discrete data. See <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html> for details.
- **filename** (*str or None (default: None)*) – file to which the output is saved

- **dpi** (*int or None, optional (default: None)*) – The resolution in dots per inch. If None it will default to the value savefig.dpi in the matplotlibrc file. If 'figure' it will set the dpi to be the value of the figure. Only used if filename is not None.
- ****plot_kwargs** (*keyword arguments*) – Extra arguments passed to *matplotlib.pyplot.scatter*.

Returns **ax** – axis on which plot was drawn

Return type *matplotlib.Axes*

Examples

```
>>> import phate
>>> import matplotlib.pyplot as plt
>>> ###
>>> # Running PHATE
>>> ###
>>> tree_data, tree_clusters = phate.tree.gen_dla(n_dim=100, n_branch=20,
...                                                branch_length=100)
>>> tree_data.shape
(2000, 100)
>>> phate_operator = phate.PHATE(k=5, a=20, t=150)
>>> tree_phate = phate_operator.fit_transform(tree_data)
>>> tree_phate.shape
(2000, 2)
>>> ###
>>> # Plotting using phate.plot
>>> ###
>>> phate.plot.scatter2d(tree_phate, c=tree_clusters)
>>> # You can also pass the PHATE operator instead of data
>>> phate.plot.scatter2d(phate_operator, c=tree_clusters)
>>> phate.plot.scatter3d(phate_operator, c=tree_clusters)
>>> ###
>>> # Using a cmap dictionary
>>> ###
>>> import numpy as np
>>> X = np.random.normal(0,1,[1000,2])
>>> c = np.random.choice(['a','b'], 1000, replace=True)
>>> X[c=='a'] += 10
>>> phate.plot.scatter2d(X, c=c, cmap={'a' : [1,0,0,1], 'b' : 'xkcd:sky blue'})
```

phate.plot.**scatter2d**(*data*, ***kwargs*)

Create a 2D scatter plot

Builds upon *matplotlib.pyplot.scatter* with nice defaults and handles categorical colors / legends better.

Parameters

- **data** (*array-like, shape=[n_samples, n_features]*) – Input data. Only the first two components will be used.
- **c** (*list-like or None, optional (default: None)*) – Color vector. Can be a single color value (RGB, RGBA, or named matplotlib colors), an array of these of length n_samples, or a list of discrete or continuous values of any data type. If *c* is not a single or list of matplotlib colors, the values in *c* will be used to populate the legend / colorbar with colors from *cmap*

- **cmap** (*matplotlib* colormap, str, dict or None, optional (default: None)) – matplotlib colormap. If None, uses *tab20* for discrete data and *inferno* for continuous data. If a dictionary, expects one key for every unique value in *c*, where values are valid matplotlib colors (hsv, rbg, rgba, or named colors)
- **s** (*float, optional (default: 1)*) – Point size.
- **discrete** (*bool or None, optional (default: None)*) – If True, the legend is categorical. If False, the legend is a colorbar. If None, discreteness is detected automatically. Data containing non-numeric *c* is always discrete, and numeric data with 20 or less unique values is discrete.
- **ax** (*matplotlib.Axes* or None, optional (default: None)) – axis on which to plot. If None, an axis is created
- **legend** (*bool, optional (default: True)*) – States whether or not to create a legend. If data is continuous, the legend is a colorbar.
- **figsize** (*tuple, optional (default: None)*) – Tuple of floats for creation of new *matplotlib* figure. Only used if *ax* is None.
- **xticks** (*True, False, or list-like (default: False)*) – If True, keeps default x ticks. If False, removes x ticks. If a list, sets custom x ticks
- **yticks** (*True, False, or list-like (default: False)*) – If True, keeps default y ticks. If False, removes y ticks. If a list, sets custom y ticks
- **zticks** (*True, False, or list-like (default: False)*) – If True, keeps default z ticks. If False, removes z ticks. If a list, sets custom z ticks. Only used for 3D plots.
- **xticklabels** (*True, False, or list-like (default: True)*) – If True, keeps default x tick labels. If False, removes x tick labels. If a list, sets custom x tick labels
- **yticklabels** (*True, False, or list-like (default: True)*) – If True, keeps default y tick labels. If False, removes y tick labels. If a list, sets custom y tick labels
- **zticklabels** (*True, False, or list-like (default: True)*) – If True, keeps default z tick labels. If False, removes z tick labels. If a list, sets custom z tick labels. Only used for 3D plots.
- **label_prefix** (*str or None (default: "PHATE")*) – Prefix for all axis labels. Axes will be labelled *label_prefix‘1, ‘label_prefix‘2, etc. Can be overriden by setting ‘xlabel*, *ylabel*, and *zlabel*.
- **xlabel** (*str or None (default : None)*) – Label for the x axis. Overrides the automatic label given by label_prefix. If None and label_prefix is None, no label is set.
- **ylabel** (*str or None (default : None)*) – Label for the y axis. Overrides the automatic label given by label_prefix. If None and label_prefix is None, no label is set.
- **zlabel** (*str or None (default : None)*) – Label for the z axis. Overrides the automatic label given by label_prefix. If None and label_prefix is None, no label is set. Only used for 3D plots.
- **title** (*str or None (default: None)*) – axis title. If None, no title is set.
- **legend_title** (*str (default: "")*) – title for the colorbar of legend

- **legend_loc** (*int or string or pair of floats, default: 'best'*) – Matplotlib legend location. Only used for discrete data. See <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html> for details.
- **filename** (*str or None (default: None)*) – file to which the output is saved
- **dpi** (*int or None, optional (default: None)*) – The resolution in dots per inch. If None it will default to the value savefig.dpi in the matplotlibrc file. If 'figure' it will set the dpi to be the value of the figure. Only used if filename is not None.
- ****plot_kwargs** (*keyword arguments*) – Extra arguments passed to *matplotlib.pyplot.scatter*.

Returns **ax** – axis on which plot was drawn

Return type *matplotlib.Axes*

Examples

```
>>> import phate
>>> import matplotlib.pyplot as plt
>>> ###
>>> # Running PHATE
>>> ###
>>> tree_data, tree_clusters = phate.tree.gen_dla(n_dim=100, n_branch=20,
...                                               branch_length=100)
>>> tree_data.shape
(2000, 100)
>>> phate_operator = phate.PHATE(k=5, a=20, t=150)
>>> tree_phate = phate_operator.fit_transform(tree_data)
>>> tree_phate.shape
(2000, 2)
>>> ###
>>> # Plotting using phate.plot
>>> ###
>>> phate.plot.scatter2d(tree_phate, c=tree_clusters)
>>> # You can also pass the PHATE operator instead of data
>>> phate.plot.scatter2d(phate_operator, c=tree_clusters)
>>> phate.plot.scatter3d(phate_operator, c=tree_clusters)
>>> ###
>>> # Using a cmap dictionary
>>> ###
>>> import numpy as np
>>> X = np.random.normal(0,1,[1000,2])
>>> c = np.random.choice(['a','b'], 1000, replace=True)
>>> X[c=='a'] += 10
>>> phate.plot.scatter2d(X, c=c, cmap={'a' : [1,0,0,1], 'b' : 'xkcd:sky blue'})
```

phate.plot.**scatter3d**(*data*, ***kwargs*)

Create a 3D scatter plot

Builds upon *matplotlib.pyplot.scatter* with nice defaults and handles categorical colors / legends better.

Parameters

- **data** (*array-like, shape=[n_samples, n_features]*) – to be the value of the figure. Only used if filename is not None. Input data. Only the first three components will be used.

- **c** (*list-like or None, optional (default: None)*) – Color vector. Can be a single color value (RGB, RGBA, or named matplotlib colors), an array of these of length n_samples, or a list of discrete or continuous values of any data type. If *c* is not a single or list of matplotlib colors, the values in *c* will be used to populate the legend / colorbar with colors from *cmap*
- **cmap** (*matplotlib* colormap, str, dict or None, optional (default: None)) – matplotlib colormap. If None, uses *tab20* for discrete data and *inferno* for continuous data. If a dictionary, expects one key for every unique value in *c*, where values are valid matplotlib colors (hsv, rbg, rgba, or named colors)
- **s** (*float, optional (default: 1)*) – Point size.
- **discrete** (*bool or None, optional (default: None)*) – If True, the legend is categorical. If False, the legend is a colorbar. If None, discreteness is detected automatically. Data containing non-numeric *c* is always discrete, and numeric data with 20 or less unique values is discrete.
- **ax** (*matplotlib.Axes* or None, optional (default: None)) – axis on which to plot. If None, an axis is created
- **legend** (*bool, optional (default: True)*) – States whether or not to create a legend. If data is continuous, the legend is a colorbar.
- **figsize** (*tuple, optional (default: None)*) – Tuple of floats for creation of new *matplotlib* figure. Only used if *ax* is None.
- **xticks** (*True, False, or list-like (default: False)*) – If True, keeps default x ticks. If False, removes x ticks. If a list, sets custom x ticks
- **yticks** (*True, False, or list-like (default: False)*) – If True, keeps default y ticks. If False, removes y ticks. If a list, sets custom y ticks
- **zticks** (*True, False, or list-like (default: False)*) – If True, keeps default z ticks. If False, removes z ticks. If a list, sets custom z ticks. Only used for 3D plots.
- **xticklabels** (*True, False, or list-like (default: True)*) – If True, keeps default x tick labels. If False, removes x tick labels. If a list, sets custom x tick labels
- **yticklabels** (*True, False, or list-like (default: True)*) – If True, keeps default y tick labels. If False, removes y tick labels. If a list, sets custom y tick labels
- **zticklabels** (*True, False, or list-like (default: True)*) – If True, keeps default z tick labels. If False, removes z tick labels. If a list, sets custom z tick labels. Only used for 3D plots.
- **label_prefix** (*str or None (default: "PHATE")*) – Prefix for all axis labels. Axes will be labelled *label_prefix‘1, ‘label_prefix‘2, etc. Can be overriden by setting ‘xlabel*, *ylabel*, and *zlabel*.
- **xlabel** (*str or None (default : None)*) – Label for the x axis. Overrides the automatic label given by label_prefix. If None and label_prefix is None, no label is set.
- **ylabel** (*str or None (default : None)*) – Label for the y axis. Overrides the automatic label given by label_prefix. If None and label_prefix is None, no label is set.
- **zlabel** (*str or None (default : None)*) – Label for the z axis. Overrides the automatic label given by label_prefix. If None and label_prefix is None, no label is set. Only used for 3D plots.

- **title** (*str or None (default: None)*) – axis title. If None, no title is set.
- **legend_title** (*str (default: "")*) – title for the colorbar of legend
- **legend_loc** (*int or string or pair of floats, default: 'best'*) – Matplotlib legend location. Only used for discrete data. See <https://matplotlib.org/api/_as_gen/matplotlib.pyplot.legend.html> for details.
- **filename** (*str or None (default: None)*) – file to which the output is saved
- **dpi** (*int or None, optional (default: None)*) – The resolution in dots per inch. If None it will default to the value savefig.dpi in the matplotlibrc file. If 'figure' it will set the dpi to be the value of the figure. Only used if filename is not None.
- ****plot_kwargs** (*keyword arguments*) – Extra arguments passed to *matplotlib.pyplot.scatter*.

Returns **ax** – axis on which plot was drawn

Return type *matplotlib.Axes*

Examples

```
>>> import phate
>>> import matplotlib.pyplot as plt
>>> ###
>>> # Running PHATE
>>> ###
>>> tree_data, tree_clusters = phate.tree.gen_dla(n_dim=100, n_branch=20,
...                                               branch_length=100)
>>> tree_data.shape
(2000, 100)
>>> phate_operator = phate.PHATE(k=5, a=20, t=150)
>>> tree_phate = phate_operator.fit_transform(tree_data)
>>> tree_phate.shape
(2000, 2)
>>> ###
>>> # Plotting using phate.plot
>>> ###
>>> phate.plot.scatter2d(tree_phate, c=tree_clusters)
>>> # You can also pass the PHATE operator instead of data
>>> phate.plot.scatter2d(phate_operator, c=tree_clusters)
>>> phate.plot.scatter3d(phate_operator, c=tree_clusters)
>>> ###
>>> # Using a cmap dictionary
>>> ###
>>> import numpy as np
>>> X = np.random.normal(0,1,[1000,2])
>>> c = np.random.choice(['a','b'], 1000, replace=True)
>>> X[c=='a'] += 10
>>> phate.plot.scatter2d(X, c=c, cmap={'a' : [1,0,0,1], 'b' : 'xkcd:sky blue'})
```

3.4 Example Data

`phate.tree.artificial_tree()`

`phate.tree.`**`gen_dla`**`(`*n_dim=100*, *n_branch=20*, *branch_length=100*, *rand_multiplier=2*, *seed=37*, *sigma=4*`)`

CHAPTER 4

Quick Start

If you have loaded a data matrix `data` in Python (cells on rows, genes on columns) you can run PHATE as follows:

```
import phate
phate_op = phate.PHATE()
data_phate = phate_op.fit_transform(data)
```

PHATE accepts the following data types: `numpy.array`, `scipy.spmatrix`, `pandas.DataFrame` and `anndata.AnnData`.

CHAPTER 5

Usage

To run PHATE on your dataset, create a PHATE operator and run *fit_transform*. Here we show an example with an artificial tree:

```
import phate
tree_data, tree_clusters = phate.tree.gen_dla()
phate_operator = phate.PHATE(k=15, t=100)
tree_phate = phate_operator.fit_transform(tree_data)
phate.plot.scatter2d(phate_operator, c=tree_clusters)
# or phate.plot.scatter2d(tree_phate, c=tree_clusters)
phate.plot.rotate_scatter3d(phate_operator, c=tree_clusters)
```

CHAPTER 6

Help

If you have any questions or require assistance using PHATE, please contact us at https://krishnaswamylab.org/get-help

class phate.**PHATE**(*n_components=2*, *knn=5*, *decay=40*, *n_landmark=2000*, *t='auto'*, *gamma=1*, *n_pca=100*, *mds_solver='sgd'*, *knn_dist='euclidean'*, *mds_dist='euclidean'*, *mds='metric'*, *n_jobs=1*, *random_state=None*, *verbose=1*, *potential_method=None*, *alpha_decay=None*, *njobs=None*, *k=None*, *a=None*, ***kwargs*)

PHATE operator which performs dimensionality reduction.

Potential of Heat-diffusion for Affinity-based Trajectory Embedding (PHATE) embeds high dimensional single-cell data into two or three dimensions for visualization of biological progressions as described in Moon et al, 2017[1].

Parameters

- **n_components** (*int, optional, default: 2*) – number of dimensions in which the data will be embedded
- **knn** (*int, optional, default: 5*) – number of nearest neighbors on which to build kernel
- **decay** (*int, optional, default: 40*) – sets decay rate of kernel tails. If None, alpha decaying kernel is not used
- **n_landmark** (*int, optional, default: 2000*) – number of landmarks to use in fast PHATE
- **t** (*int, optional, default: 'auto'*) – power to which the diffusion operator is powered. This sets the level of diffusion. If 'auto', t is selected according to the knee point in the Von Neumann Entropy of the diffusion operator
- **gamma** (*float, optional, default: 1*) – Informational distance constant between -1 and 1. *gamma=1* gives the PHATE log potential, *gamma=0* gives a square root potential.

[1] Moon KR, van Dijk D, Zheng W, *et al.* (2017), *PHATE: A Dimensionality Reduction Method for Visualizing Trajectory Structures in High-Dimensional Biological Data*, BioRxiv.

- **n_pca** (*int, optional, default: 100*) – Number of principal components to use for calculating neighborhoods. For extremely large datasets, using n_pca < 20 allows neighborhoods to be calculated in roughly log(n_samples) time.
- **mds_solver** (*{'sgd', 'smacof'}, optional (default: 'sgd')*) – which solver to use for metric MDS. SGD is substantially faster, but produces slightly less optimal results. Note that SMACOF was used for all figures in the PHATE paper.
- **knn_dist** (*string, optional, default: 'euclidean'*) – recommended values: 'euclidean', 'cosine', 'precomputed' Any metric from *scipy.spatial.distance* can be used distance metric for building kNN graph. Custom distance functions of form *f(x, y) = d* are also accepted. If 'precomputed', *data* should be an n_samples x n_samples distance or affinity matrix. Distance matrices are assumed to have zeros down the diagonal, while affinity matrices are assumed to have non-zero values down the diagonal. This is detected automatically using *data[0,0]*. You can override this detection with *knn_dist='precomputed_distance'* or *knn_dist='precomputed_affinity'*.
- **mds_dist** (*string, optional, default: 'euclidean'*) – Distance metric for MDS. Recommended values: 'euclidean' and 'cosine' Any metric from *scipy.spatial.distance* can be used. Custom distance functions of form *f(x, y) = d* are also accepted
- **mds** (*string, optional, default: 'metric'*) – choose from ['classic', 'metric', 'nonmetric']. Selects which MDS algorithm is used for dimensionality reduction
- **n_jobs** (*integer, optional, default: 1*) – The number of jobs to use for the computation. If -1 all CPUs are used. If 1 is given, no parallel computing code is used at all, which is useful for debugging. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used. Thus for n_jobs = -2, all CPUs but one are used
- **random_state** (*integer or numpy.RandomState, optional, default: None*) – The generator used to initialize SMACOF (metric, nonmetric) MDS If an integer is given, it fixes the seed Defaults to the global *numpy* random number generator
- **verbose** (*int* or *boolean*, optional (default: 1)) – If *True* or > *0*, print status messages
- **potential_method** (*deprecated.*) – Use *gamma=1* for log transformation and *gamma=0* for square root transformation.
- **alpha_decay** (*deprecated.*) – Use *decay=None* to disable alpha decay
- **njobs** (*deprecated.*) – Use n_jobs to match *sklearn* standards
- **k** (Deprecated for *knn*) –
- **a** (Deprecated for *decay*) –
- **kwargs** (additional arguments for *graphtools.Graph*) –

X

Type array-like, shape=[n_samples, n_dimensions]

embedding

Stores the position of the dataset in the embedding space

Type array-like, shape=[n_samples, n_components]

diff_op

The diffusion operator built from the graph

Type array-like, shape=[n_samples, n_samples] or [n_landmark, n_landmark]

graph
The graph built on the input data

Type graphtools.base.BaseGraph

optimal_t
The automatically selected t, when t = 'auto'. When t is given, optimal_t is None.

Type int

Examples

```
>>> import phate
>>> import matplotlib.pyplot as plt
>>> tree_data, tree_clusters = phate.tree.gen_dla(n_dim=100, n_branch=20,
...                                               branch_length=100)
>>> tree_data.shape
(2000, 100)
>>> phate_operator = phate.PHATE(knn=5, decay=20, t=150)
>>> tree_phate = phate_operator.fit_transform(tree_data)
>>> tree_phate.shape
(2000, 2)
>>> phate.plot.scatter2d(tree_phate, c=tree_clusters)
```

References

diff_op
The diffusion operator calculated from the data

diff_potential
Interpolates the PHATE potential to one entry per cell

This is equivalent to calculating infinite-dimensional PHATE, or running PHATE without the MDS step.

Returns **diff_potential**

Return type ndarray, shape=[n_samples, min(n_landmark, n_samples)]

fit (*X*)
Computes the diffusion operator

Parameters **X** (*array, shape=[n_samples, n_features]*) – input data with *n_samples* samples and *n_dimensions* dimensions. Accepted data types: *numpy.ndarray*, *scipy.sparse.spmatrix*, *pd.DataFrame*, *anndata.AnnData*. If *knn_dist* is 'precomputed', *data* should be a n_samples x n_samples distance or affinity matrix

Returns

- **phate_operator** (*PHATE*)
- *The estimator object*

fit_transform (*X*, ***kwargs*)
Computes the diffusion operator and the position of the cells in the embedding space

Parameters

- **X** (*array, shape=[n_samples, n_features]*) – input data with *n_samples* samples and *n_dimensions* dimensions. Accepted data types: *numpy.ndarray*, *scipy.sparse.spmatrix*, *pd.DataFrame*, *anndata.AnnData* If *knn_dist* is 'precomputed', *data* should be a n_samples x n_samples distance or affinity matrix

- **kwargs** (further arguments for *PHATE.transform()*) – Keyword arguments as specified in `transform()`

Returns embedding – The cells embedded in a lower dimensional space using PHATE

Return type array, shape=[n_samples, n_dimensions]

reset_mds(***kwargs*)

Deprecated. Reset parameters related to multidimensional scaling

Parameters

- **n_components** (*int, optional, default: None*) – If given, sets number of dimensions in which the data will be embedded
- **mds** (*string, optional, default: None*) – choose from ['classic', 'metric', 'nonmetric'] If given, sets which MDS algorithm is used for dimensionality reduction
- **mds_dist** (*string, optional, default: None*) – recommended values: 'euclidean' and 'cosine' Any metric from scipy.spatial.distance can be used If given, sets the distance metric for MDS

reset_potential(***kwargs*)

Deprecated. Reset parameters related to the diffusion potential

Parameters

- **t** (*int or 'auto', optional, default: None*) – Power to which the diffusion operator is powered If given, sets the level of diffusion
- **potential_method** (*string, optional, default: None*) – choose from ['log', 'sqrt'] If given, sets which transformation of the diffusional operator is used to compute the diffusion potential

set_params(***params*)

Set the parameters on this estimator.

Any parameters not given as named arguments will be left at their current value.

Parameters

- **n_components** (*int, optional, default: 2*) – number of dimensions in which the data will be embedded
- **knn** (*int, optional, default: 5*) – number of nearest neighbors on which to build kernel
- **decay** (*int, optional, default: 40*) – sets decay rate of kernel tails. If None, alpha decaying kernel is not used
- **n_landmark** (*int, optional, default: 2000*) – number of landmarks to use in fast PHATE
- **t** (*int, optional, default: 'auto'*) – power to which the diffusion operator is powered. This sets the level of diffusion. If 'auto', t is selected according to the knee point in the Von Neumann Entropy of the diffusion operator
- **gamma** (*float, optional, default: 1*) – Informational distance constant between -1 and 1. *gamma=1* gives the PHATE log potential, *gamma=0* gives a square root potential.

- **n_pca** (*int, optional, default: 100*) – Number of principal components to use for calculating neighborhoods. For extremely large datasets, using n_pca < 20 allows neighborhoods to be calculated in roughly log(n_samples) time.
- **mds_solver** (*{'sgd', 'smacof'}, optional (default: 'sgd')*) – which solver to use for metric MDS. SGD is substantially faster, but produces slightly less optimal results. Note that SMACOF was used for all figures in the PHATE paper.
- **knn_dist** (*string, optional, default: 'euclidean'*) – recommended values: 'euclidean', 'cosine', 'precomputed' Any metric from *scipy.spatial.distance* can be used distance metric for building kNN graph. Custom distance functions of form *f(x, y) = d* are also accepted. If 'precomputed', *data* should be an n_samples x n_samples distance or affinity matrix. Distance matrices are assumed to have zeros down the diagonal, while affinity matrices are assumed to have non-zero values down the diagonal. This is detected automatically using *data[0,0]*. You can override this detection with *knn_dist='precomputed_distance'* or *knn_dist='precomputed_affinity'*.
- **mds_dist** (*string, optional, default: 'euclidean'*) – recommended values: 'euclidean' and 'cosine' Any metric from *scipy.spatial.distance* can be used distance metric for MDS
- **mds** (*string, optional, default: 'metric'*) – choose from ['classic', 'metric', 'nonmetric']. Selects which MDS algorithm is used for dimensionality reduction
- **n_jobs** (*integer, optional, default: 1*) – The number of jobs to use for the computation. If -1 all CPUs are used. If 1 is given, no parallel computing code is used at all, which is useful for debugging. For n_jobs below -1, (n_cpus + 1 + n_jobs) are used. Thus for n_jobs = -2, all CPUs but one are used
- **random_state** (*integer or numpy.RandomState, optional, default: None*) – The generator used to initialize SMACOF (metric, nonmetric) MDS If an integer is given, it fixes the seed Defaults to the global *numpy* random number generator
- **verbose** (*int* or *boolean*, optional (default: 1)) – If *True* or > *0*, print status messages
- **k** (Deprecated for *knn*) –
- **a** (Deprecated for *decay*) –

Examples

```
>>> import phate
>>> import matplotlib.pyplot as plt
>>> tree_data, tree_clusters = phate.tree.gen_dla(n_dim=50, n_branch=5,
...                                               branch_length=50)
>>> tree_data.shape
(250, 50)
>>> phate_operator = phate.PHATE(k=5, a=20, t=150)
>>> tree_phate = phate_operator.fit_transform(tree_data)
>>> tree_phate.shape
(250, 2)
>>> phate_operator.set_params(n_components=10)
PHATE(a=20, alpha_decay=None, k=5, knn_dist='euclidean', mds='metric',
   mds_dist='euclidean', n_components=10, n_jobs=1, n_landmark=2000,
   n_pca=100, njobs=None, potential_method='log', random_state=None, t=150,
   verbose=1)
```

(continues on next page)

(continued from previous page)

```
>>> tree_phate = phate_operator.transform()
>>> tree_phate.shape
(250, 10)
>>> # plt.scatter(tree_phate[:,0], tree_phate[:,1], c=tree_clusters)
>>> # plt.show()
```

Returns

Return type self

`transform`(*X=None*, *t_max=100*, *plot_optimal_t=False*, *ax=None*)

Computes the position of the cells in the embedding space

Parameters

- **X** (*array, optional, shape=[n_samples, n_features]*) – input data with *n_samples* samples and *n_dimensions* dimensions. Not required, since PHATE does not currently embed cells not given in the input matrix to *PHATE.fit()*. Accepted data types: *numpy.ndarray*, *scipy.sparse.spmatrix*, *pd.DataFrame*, *anndata.AnnData*. If *knn_dist* is 'precomputed', *data* should be a n_samples x n_samples distance or affinity matrix
- **t_max** (*int, optional, default: 100*) – maximum t to test if *t* is set to 'auto'
- **plot_optimal_t** (*boolean, optional, default: False*) – If true and *t* is set to 'auto', plot the Von Neumann entropy used to select t
- **ax** (*matplotlib.axes.Axes, optional*) – If given and *plot_optimal_t* is true, plot will be drawn on the given axis.

Returns

- **embedding** (*array, shape=[n_samples, n_dimensions]*)
- *The cells embedded in a lower dimensional space using PHATE*

Python Module Index

Index