# Contents

1 Installation
   1.1 Python installation ................................................................. 3
   1.2 MATLAB installation ............................................................... 3
   1.3 R installation ............................................................................. 4

2 Tutorial ......................................................................................... 5

3 API
   3.1 MAGIC ....................................................................................... 7
   3.2 Plotting ...................................................................................... 11

4 Quick Start .................................................................................... 13

5 Help ............................................................................................... 15

Python Module Index ......................................................................... 21
Markov Affinity-based Graph Imputation of Cells (MAGIC) is an algorithm for denoising and imputation of single cells applied to single-cell RNA sequencing data. To see how MAGIC can be applied to single-cell RNA-seq, elucidating the epithelial-to-mesenchymal transition, read our publication in Cell.

1.1 Python installation

1.1.1 Installation with pip

The Python version of MAGIC can be installed using:

```
pip install --user magic-impute
```

1.1.2 Installation from source

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

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

1.2 MATLAB installation

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

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

2. Add the MAGIC/Matlab directory to your MATLAB path and run any of our run or test scripts to get a feel for MAGIC.
1.3 R installation

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

If `python` or `pip` are not installed, you will need to install them. We recommend Miniconda3 to install Python and `pip` together, or otherwise you can install `pip` from https://pip.pypa.io/en/stable/installing/.

1.3.1 Installation from CRAN

In R, run this command to install MAGIC and all dependencies:

```r
install.packages("Rmagic")
```

In a terminal, run the following command to install the Python repository:

```bash
pip install --user magic-impute
```

1.3.2 Installation from source

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

```bash
git clone https://github.com/KrishnaswamyLab/MAGIC.git
cd MAGIC/Rmagic
R CMD INSTALL .
cd ../python
python setup.py install --user
```

If the `Rmagic` 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:

```bash
cd MAGIC
git submodule init
git submodule update
cd Rmagic
R CMD INSTALL
cd ../python
python setup.py install --user
To run MAGIC on your dataset, create a MAGIC operator and run `fit_transform`. Here we show an example with an artificial test dataset located in the MAGIC repository:

```python
import magic
import matplotlib.pyplot as plt
import pandas as pd
X = pd.read_csv("MAGIC/data/test_data.csv")
magic_operator = magic.MAGIC()
X_magic = magic_operator.fit_transform(X, genes=['VIM', 'CDH1', 'ZEB1'])
plt.scatter(X_magic['VIM'], X_magic['CDH1'], c=X_magic['ZEB1'], s=1, cmap='inferno')
plt.show()
magic.plot.animate_magic(X, gene_x='VIM', gene_y='CDH1', gene_color='ZEB1', operator=magic_operator)
```

A demo on MAGIC usage for single cell RNA-seq data can be found in this notebook: http://nbviewer.jupyter.org/github/KrishnaswamyLab/magic/blob/master/python/tutorial_notebooks/emt_tutorial.ipynb

A second tutorial analyzing myeloid and erythroid cells in mouse bone marrow is available here: http://nbviewer.jupyter.org/github/KrishnaswamyLab/magic/blob/master/python/tutorial_notebooks/bonemarrow_tutorial.ipynb
3.1 MAGIC

Markov Affinity-based Graph Imputation of Cells (MAGIC)

Authors: Scott Gigante <scott.gigante@yale.edu>, Daniel Dager <daniel.dager@yale.edu> (C) 2018 Krishnaswamy Lab GPLv2

class magic.magic.MAGIC (k=10, a=15, t='auto', n_pca=100, knn_dist='euclidean', n_jobs=1, random_state=None, verbose=1)

Bases: sklearn.base.BaseEstimator

MAGIC operator which performs dimensionality reduction.

Markov Affinity-based Graph Imputation of Cells (MAGIC) is an algorithm for denoising and transcript recover of single cells applied to single-cell RNA sequencing data, as described in van Dijk et al, 2018\(^1\).

Parameters

- **k** (int, optional, default: 10) – number of nearest neighbors on which to build kernel
- **a** (int, optional, default: 15) – sets decay rate of kernel tails. If None, alpha decaying kernel is not used
- **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 Procrustes disparity of the diffused data
- **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.
- **knn_dist** (string, optional, default: 'euclidean') – recommended values: ‘euclidean’, ‘cosine’ Any metric from scipy.spatial.distance can be used distance metric for building kNN graph.

---

\(^1\) Van Dijk D et al. (2018), Recovering Gene Interactions from Single-Cell Data Using Data Diffusion, Cell.
• **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 random PCA 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

---

**X**

array-like, shape=[n_samples, n_features] – Input data

**X_magic**

array-like, shape=[n_samples, n_features] – Output data

**graph**

graphtools.BaseGraph – The graph built on the input data

---

**Examples**

```python
>>> import magic
>>> import pandas as pd
>>> import matplotlib.pyplot as plt

>>> X = pd.read_csv("../../data/test_data.csv")
>>> X.shape
(500, 197)
>>> magic_operator = magic.MAGIC()
>>> X_magic = magic_operator.fit_transform(X, genes=['VIM', 'CDH1', 'ZEB1'])
>>> X_magic.shape
(500, 3)
>>> magic_operator.set_params(t=7)
MAGIC(a=15, k=5, knn_dist='euclidean', n_jobs=1, n_pca=100,
    random_state=None, t=7, verbose=1)
>>> X_magic = magic_operator.transform(genes=['VIM', 'CDH1', 'ZEB1'])
>>> X_magic.shape
(500, 3)
>>> X_magic = magic_operator.transform(genes="all_genes")
>>> X_magic.shape
(500, 197)
>>> plt.scatter(X_magic['VIM'], X_magic['CDH1'],
    ... c=X_magic['ZEB1'], s=1, cmap='inferno')
>>> plt.show()
``` 

---

**References**

**diff_op**

The diffusion operator calculated from the data

**fit**(X, graph=None)

Computes the diffusion operator

**Parameters**
• **X**: (array, shape=[n_samples, n_features]) – input data with n_samples samples and n_features dimensions. Accepted data types: numpy.ndarray, scipy.sparse.spmatrix, pd.DataFrame, anndata.AnnData.

• **graph** (graphtools.Graph, optional (default: None)) – If given, provides a precomputed kernel matrix with which to perform diffusion.

Returns magic_operator – The estimator object

Return type **MAGIC**

**fit_transform**(X, graph=None, **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_features dimensions. Accepted data types: numpy.ndarray, scipy.sparse.spmatrix, pd.DataFrame, anndata.AnnData.

• **graph** (graphtools.Graph, optional (default: None)) – If given, provides a precomputed kernel matrix with which to perform diffusion.

• **genes** (list or "all_genes", "pca_only", optional (default: None)) – List of genes, either as integer indices or column names if input data is a pandas DataFrame. If “all_genes”, the entire smoothed matrix is returned. If “pca_only”, PCA on the smoothed data is returned. If None, the entire matrix is also returned, but a warning may be raised if the resultant matrix is very large.

• **t_max** (int, optional, default: 20) – 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 disparity 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 X_magic – The gene expression values after diffusion

Return type array, shape=[n_samples, n_genes]

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

Parameters **deep** (boolean, optional) – 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

**knnDREMI**(gene_x, gene_y, k=10, n_bins=20, n_mesh=3, n_jobs=1, plot=False, **kwargs)
Calculate kNN-DREMI on MAGIC output

Calculates k-Nearest Neighbor conditional Density Resampled Estimate of Mutual Information as defined in Van Dijk et al, 2018.¹

Note that kNN-DREMI, like Mutual Information and DREMI, is not symmetric. Here we are estimating I(Y|X).

Parameters

• **gene_x**: (array-like, shape=[n_samples]) – Gene shown on the x axis (independent feature)
• **gene_y** *(array-like, shape=[n_samples])* – Gene shown on the y axis (dependent feature)

• **k** *(int, range=[0:n_samples], optional (default: 10)) – Number of neighbors

• **n_bins** *(int, range=[0:inf), optional (default: 20)) – Number of bins for density resampling

• **n_mesh** *(int, range=[0:inf), optional (default: 3)) – In each bin, density will be calculated around (mesh ** 2) points

• **n_jobs** *(int, optional (default: 1)) – Number of threads used for kNN calculation

• **plot** *(bool, optional (default: False)) – If True, DREMI create plots of the data like those seen in Fig 5C/D of van Dijk et al. 2018. (doi:10.1016/j.cell.2018.05.061).

• ****kwargs **(additional arguments for scprep.stats.plot_knnDREMI) –**

**Returns** dremi – kNN conditional Density resampled estimate of mutual information

**Return type** float

set_params(**params**)

Set the parameters on this estimator.

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

**Parameters**

• **k** *(int, optional, default: 10) – number of nearest neighbors on which to build kernel

• **a** *(int, optional, default: 15) – sets decay rate of kernel tails. If None, alpha decaying kernel is not used

• **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 R squared of the diffused data

• **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.

• **knn_dist** *(string, optional, default: 'euclidean') – recommended values: ‘euclidean’, ‘cosine’ Any metric from scipy.spatial.distance can be used distance metric for building kNN graph.

• **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 random PCA 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

**Returns**

**Return type** self
transform \((X=None, \text{genes}=\text{None}, t\_max=20, \text{plot\_optimal\_t}=\text{False}, ax=\text{None})\)

Computes the values of genes after diffusion

**Parameters**

- \(X\) (array, optional, shape=[n\_samples, n\_features]) – input data with \(n\_samples\) samples and \(n\_features\) dimensions. Not required, since MAGIC does not embed cells not given in the input matrix to MAGIC.fit(). Accepted data types: numpy.ndarray, scipy.sparse.spmatrix, pd.DataFrame, anndata.AnnData.

- \(\text{genes}\) (list or {"all\_genes", "pca\_only"}, optional (default: None)) – List of genes, either as integer indices or column names if input data is a pandas DataFrame. If “all\_genes”, the entire smoothed matrix is returned. If “pca\_only”, PCA on the smoothed data is returned. If None, the entire matrix is also returned, but a warning may be raised if the resultant matrix is very large.

- \(t\_max\) (int, optional, default: 20) – maximum t to test if \(t\) is set to ‘auto’

- \(\text{plot\_optimal\_t}\) (boolean, optional, default: False) – If true and \(t\) is set to ‘auto’, plot the disparity used to select \(t\)

- \(ax\) (matplotlib.axes.Axes, optional) – If given and \(\text{plot\_optimal\_t}\) is true, plot will be drawn on the given axis.

**Returns** \(X\_magic\) – The gene expression values after diffusion

**Return type** array, shape=[\(n\_samples, n\_genes\)]

### 3.2 Plotting

magic.plot.animate_magic \((data, \text{gene\_x}, \text{gene\_y}, \text{gene\_color}=\text{None}, t\_max=20, \text{operator}=\text{None}, \text{filename}=\text{None}, ax=\text{None}, \text{figsize}=\text{None}, s=1, \text{cmap}=\text{‘inferno’}, \text{interval}=200, \text{dpi}=100, \text{ipython\_html}=\text{‘jshtml’}, \text{verbose}=\text{False}, **\text{kwargs}$$)

Animate a gene-gene relationship with increased diffusion

**Parameters**

- \(\text{data}\) (array-like) – Input data matrix

- \(\text{gene\_x}\) (int or str) – Gene to put on the x axis

- \(\text{gene\_y}\) (int or str) – Gene to put on the y axis

- \(\text{gene\_color}\) (int or str, optional (default: None)) – Gene to color by. If None, no color vector is used

- \(t\_max\) (int, optional (default: 20)) – maximum value of \(t\) to include in the animation

- \(\text{operator}\) (magic.MAGIC, optional (default: None)) – precomputed MAGIC operator. If None, one is created.

- \(\text{filename}\) (str, optional (default: None)) – If not None, saves a .gif or .mp4 with the output

- \(ax\) (matplotlib.Axes or None, optional (default: None)) – axis on which to plot. If None, an axis is created

- \(\text{figsize}\) (tuple, optional (default: None)) – Tuple of floats for creation of new matplotlib figure. Only used if \(ax\) is None.

- \(s\) (int, optional (default: 1)) – Point size
- **cmap** *(str or callable, optional (default: 'inferno'))* – Matplotlib colormap
- **interval** *(float, optional (default: 30))* – Time in milliseconds between frames
- **dpi** *(int, optional (default: 100))* – Dots per inch (image quality) in saved animation
- **ipython_html** *(('html5', 'jshtml'))* – which html writer to use if using a Jupyter Notebook
- **verbose** *(bool, optional (default: False))* – MAGIC operator verbosity
- **kwaargs** *(arguments for MAGIC)*

**Returns**

**Return type** A Matplotlib animation showing diffusion of an edge with increased t
CHAPTER 4

Quick Start

To run MAGIC on your dataset, create a MAGIC operator and run `fit_transform`. Here we show an example with a small, artificial dataset located in the MAGIC repository:

```python
import magic
import pandas as pd
import matplotlib.pyplot as plt

X = pd.read_csv("MAGIC/data/test_data.csv")
magic_operator = magic.MAGIC()
X_magic = magic_operator.fit_transform(X, genes=['VIM', 'CDH1', 'ZEB1'])
plt.scatter(X_magic['VIM'], X_magic['CDH1'], c=X_magic['ZEB1'], s=1, cmap='inferno')
plt.show()
magic.plot.animate_magic(X, gene_x='VIM', gene_y='CDH1', gene_color='ZEB1',
 operator=magic_operator)
```
If you have any questions or require assistance using MAGIC, please contact us at https://krishnaswamylab.org/get-help

```python
class magic.MAGIC(k=10, a=15, t='auto', n_pca=100, knn_dist='euclidean', n_jobs=1, random_state=None, verbose=1)
```

MAGIC operator which performs dimensionality reduction.

Markov Affinity-based Graph Imputation of Cells (MAGIC) is an algorithm for denoising and transcript recover of single cells applied to single-cell RNA sequencing data, as described in van Dijk et al, 2018.

**Parameters**

- `k (int, optional, default: 10)` – number of nearest neighbors on which to build kernel
- `a (int, optional, default: 15)` – sets decay rate of kernel tails. If None, alpha decaying kernel is not used
- `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 Procrustes disparity of the diffused data
- `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.
- `knn_dist (string, optional, default: 'euclidean')` – recommended values: ‘euclidean’, ‘cosine’ Any metric from scipy.spatial.distance can be used distance metric for building kNN graph.
- `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 random PCA. 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.

  - X
    array-like, shape=[n_samples, n_features] – Input data.
  - X_magic
    array-like, shape=[n_samples, n_features] – Output data.
  - graph
    graphtools.BaseGraph – The graph built on the input data.

**Examples**

```python
>>> import magic
>>> import pandas as pd
>>> import matplotlib.pyplot as plt
>>> X = pd.read_csv("./data/test_data.csv")
>>> X.shape
(500, 197)
>>> magic_operator = magic.MAGIC()
>>> X_magic = magic_operator.fit_transform(X, genes=['VIM', 'CDH1', 'ZEB1'])
>>> X_magic.shape
(500, 3)
>>> magic_operator.set_params(t=7)
MAGIC(a=15, k=5, knn_dist='euclidean', n_jobs=1, n_pca=100,
  random_state=None, t=7, verbose=1)
>>> X_magic = magic_operator.transform(genes=['VIM', 'CDH1', 'ZEB1'])
>>> X_magic.shape
(500, 3)
>>> X_magic = magic_operator.transform(genes="all_genes")
>>> X_magic.shape
(500, 197)
>>> plt.scatter(X_magic['VIM'], X_magic['CDH1'],
... c=X_magic['ZEB1'], s=1, cmap='inferno')
>>> plt.show()
>>> dremi = magic_operator.knnDREMI('VIM', 'CDH1', plot=True)
```

**References**

- diff_op
  The diffusion operator calculated from the data.

- fit (X, graph=None)
  Computes the diffusion operator.

  Parameters

  - X
    (array, shape=[n_samples, n_features]) – input data with n_samples samples and n_features dimensions. Accepted data types: numpy.ndarray, scipy.sparse.spmatrix, pd.DataFrame, anndata.AnnData.
• **graph** (`graphtools.Graph`, optional (default: None)) – If given, provides a precomputed kernel matrix with which to perform diffusion.

**Returns** magic_operator – The estimator object

**Return type** MAGIC

**fit_transform**(X, graph=None, **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_features dimensions. Accepted data types: numpy.ndarray, scipy.sparse.spmatrix, pd.DataFrame, anndata.AnnData.

• **graph** (`graphtools.Graph`, optional (default: None)) – If given, provides a precomputed kernel matrix with which to perform diffusion.

• **genes** (list or {"all_genes", "pca_only"}, optional (default: None)) – List of genes, either as integer indices or column names if input data is a pandas DataFrame. If “all_genes”, the entire smoothed matrix is returned. If “pca_only”, PCA on the smoothed data is returned. If None, the entire matrix is also returned, but a warning may be raised if the resultant matrix is very large.

• **t_max** (int, optional, default: 20) – 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 disparity 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** X_magic – The gene expression values after diffusion

**Return type** array, shape=[n_samples, n_genes]

**knnDREMI**(gene_x, gene_y, k=10, n_bins=20, n_mesh=3, n_jobs=1, plot=False, **kwargs)

Calculate kNN-DREMI on MAGIC output

Calculates k-Nearest Neighbor conditional Density Resampled Estimate of Mutual Information as defined in Van Dijk et al, 2018.¹

Note that kNN-DREMI, like Mutual Information and DREMI, is not symmetric. Here we are estimating I(Y|X).

**Parameters**

• **gene_x** (array-like, shape=[n_samples]) – Gene shown on the x axis (independent feature)

• **gene_y** (array-like, shape=[n_samples]) – Gene shown on the y axis (dependent feature)

• **k** (int, range=[0:n_samples), optional (default: 10)) – Number of neighbors

• **n_bins** (int, range=[0:inf), optional (default: 20)) – Number of bins for density resampling

• **n_mesh** (int, range=[0:inf), optional (default: 3)) – In each bin, density will be calculated around (mesh ** 2) points

• **n_jobs** (int, optional (default: 1)) – Number of threads used for kNN calculation
• **plot** *(bool, optional (default: False)) – If True, DREMI creates plots of the data like those seen in Fig 5C/D of van Dijk et al. 2018. (doi:10.1016/j.cell.2018.05.061).*

• **kwargs** *(additional arguments for scprep.stats.plot_knnDREMI) –*

**Returns**  
dremi – kNN conditional Density resampled estimate of mutual information

**Return type**  
float

**set_params** *(**params)*

Set the parameters on this estimator.

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

**Parameters**

- **k** *(int, optional, default: 10)* – number of nearest neighbors on which to build kernel

- **a** *(int, optional, default: 15)* – sets decay rate of kernel tails. If None, alpha decaying kernel is not used

- **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 R squared of the diffused data

- **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.

- **knn_dist** *(string, optional, default: ‘euclidean’) – recommended values: ‘euclidean’, ‘cosine’ Any metric from scipy.spatial.distance can be used distance metric for building kNN graph.

- **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 random PCA If an integer is given, it fixes the seed Default to the global numpy random number generator

- **verbose** *(int or boolean, optional (default: 1)) – If True or > 0, print status messages

**Returns**

**Return type**  
self

**transform** *(X=None, genes=None, t_max=20, plot_optimal_t=False, ax=None)*

Computes the values of genes after diffusion

**Parameters**

- **X** *(array, optional, shape=[n_samples, n_features]) – input data with n_samples samples and n_features dimensions. Not required, since MAGIC does not embed cells not given in the input matrix to MAGIC.fit(). Accepted data types: numpy.ndarray, scipy.sparse.spmatrix, pd.DataFrame, anndata.AnnData.

- **genes** *(list or {"all_genes", "pca_only"}, optional (default: None)) – List of genes, either as integer indices or column names if input data is a pandas DataFrame. If “all_genes”, the entire smoothed matrix is returned. If “pca_only”, PCA on
the smoothed data is returned. If None, the entire matrix is also returned, but a warning may be raised if the resultant matrix is very large.

- \texttt{t\_max} (\texttt{int}, \texttt{optional}, \texttt{default}: 20) – maximum t to test if \( t \) is set to ‘auto’
- \texttt{plot\_optimal\_t} (\texttt{boolean}, \texttt{optional}, \texttt{default}: \texttt{False}) – If true and \( t \) is set to ‘auto’, plot the disparity used to select \( t \)
- \texttt{ax} (\texttt{matplotlib.axes.Axes}, \texttt{optional}) – If given and \texttt{plot\_optimal\_t} is true, plot will be drawn on the given axis.

Returns \( X\_magic \) – The gene expression values after diffusion

Return type array, shape=[\text{n\_samples, n\_genes}]
Python Module Index

m
magic.magic, 7
magic.plot, 11
Index

A
animate_magic() (in module magic.plot), 11

D
diff_op (magic.magic.MAGIC attribute), 8

F
fit() (magic.magic.MAGIC method), 8
fit_transform() (magic.magic.MAGIC method), 9

G
get_params() (magic.magic.MAGIC method), 9
graph (magic.MAGIC attribute), 16
graph (magic.magic.MAGIC attribute), 8

K
knnDREMI() (magic.magic.MAGIC method), 9

M
MAGIC (class in magic.magic), 7
magic.magic (module), 7
magic.plot (module), 11

S
set_params() (magic.magic.MAGIC method), 10

T
transform() (magic.magic.MAGIC method), 10

X
X (magic.MAGIC attribute), 16
X (magic.magic.MAGIC attribute), 8
X_magic (magic.magic.MAGIC attribute), 16
X_magic (magic.magic.MAGIC attribute), 8