pydaddy

Release 1.0.0

Ashwin Karichannavar, Arshed Nabeel

Sep 25, 2023

CONTENTS

PyDaddy is a comprehensive and easy to use python package to discover data-derived stochastic differential equations from time series data. PyDaddy takes the time series of state variable x, scalar or 2-dimensional vector, as input and discovers an SDE of the form:

$$\frac{dx}{dt} = f(x) + g(x) \cdot \eta(t)$$

where $\eta(t)$ is uncorrelated white noise. The function f is called the *drift*, and governs the deterministic part of the dynamics. g^2 is called the *diffusion* and governs the stochastic part of the dynamics.

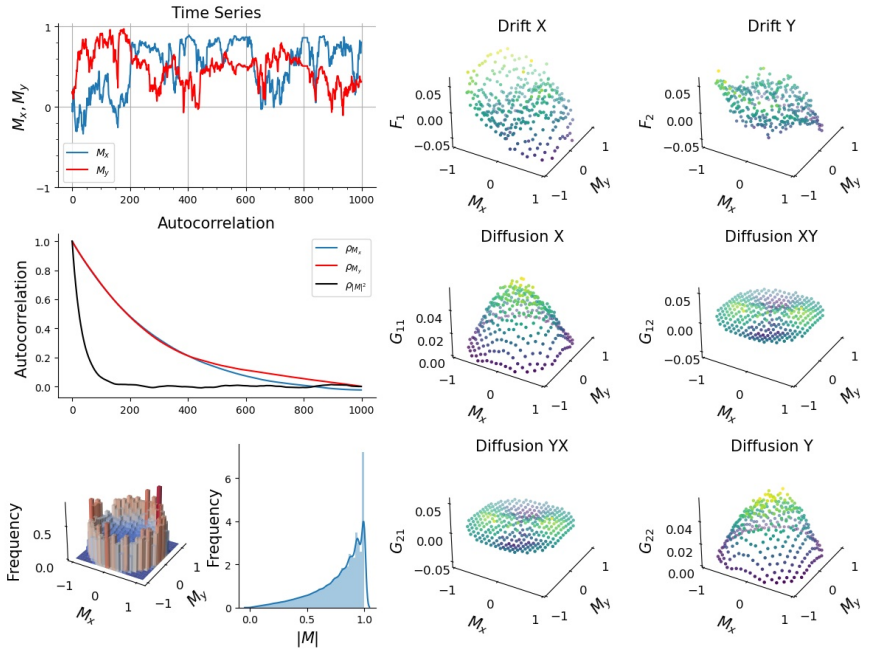

Fig. 1: An example summary plot generated by PyDaddy, for a vector time series dataset.

PyDaddy also provides a range of functionality such as equation-learning for the drift and diffusion functions using sparse regresssion and a suite of diagnostic functions.

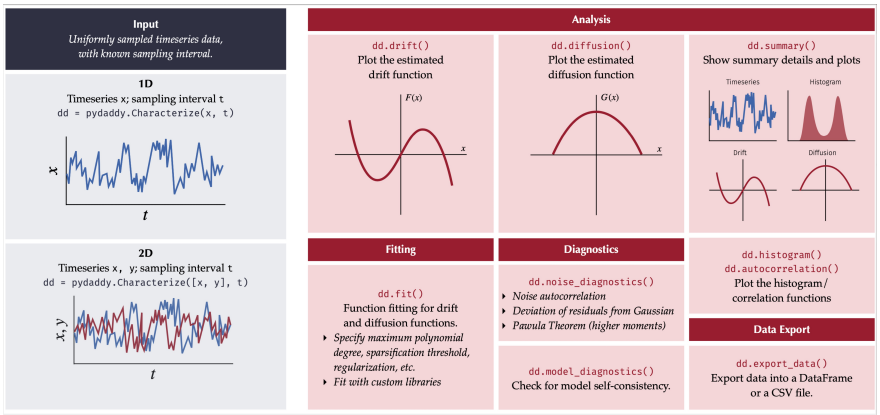

Fig. 2: Schematic illustration of PyDaddy functionality.

CHAPTER

ONE

GET STARTED

- To take PyDaddy for a walk, see the tutorial notebooks. The notebooks can be executed online on Google Colab; no installation is necessary!
- To install PyDaddy on your system, see *installation instructions*.
- See the *usage guide* and *advanced usage tips* for detailed instructions and tips on how to use PyDaddy.

CHAPTER

TWO

CITATION

If you are using this package in your research, please cite the repository and the associated paper as follows:

Nabeel, A., Karichannavar, A., Palathingal, S., Jhawar, J., Danny Raj, M., & Guttal, V. (2022). PyDaddy: A Python Package for Discovering SDEs from Time Series Data (Version 0.1.5) [Computer software]. https://github.com/tee-lab/PyDaddy

Nabeel, A., Karichannavar, A., Palathingal, S., Jhawar, J., Danny Raj, M., & Guttal, V. (2022). PyDaddy: A Python package for discovering stochastic dynamical equations from timeseries data. arXiv preprint arXiv:2205.02645.

CHAPTER

THREE

CONTENTS

3.1 Tutorials (no installation required)

The following tutorial notebooks will familiarize you with the various functionalities of PyDaddy. These notebooks can be executed on Google Colab by clicking the 'Open in Colab' buttons (**no installation required**).

Note: While executing the tutorial notebooks on Colab, do not forget to execute the cell saying:

```
%pip install git+https://github.com/tee-lab/PyDaddy.git
```

This sets up PyDaddy on your Colab environment.

You can also download these notebooks from the GitHub repo.. To download the notebooks, click on the buttons to open the notebook. Right-click on the 'Raw' button to and click 'Save Linked File…' to save the file to your computer.

3.1.1 Getting started with PyDaddy

This notebook introduces the basic functionalities of PyDaddy, using a 1-dimensional dataset. The notebook explores how to visually inspect drift and diffusion functions, how to fit analytical expressions to them, and how to use the various diagnostic tools provided.

3.1.2 Getting started with vector data

PyDaddy also works with 2-dimensional vector data. This notebook demonstrates PyDaddy operation with a vector dataset.

3.1.3 Recovering SDEs from synthetic time series

This notebook generates a simulated time series from a user-specified SDE, and uses PyDaddy to recover the drift and diffusion functions from the simulated time series. Use this notebook to play around with the PyDaddy fitting procedure and gain insights.

3.1.4 Advanced function fitting

PyDaddy can discover analytical expressions for the drift and diffusion functions. This notebook describes how to customize the fitting procedure to obtain best results.

3.1.5 Exporting data

This notebook demonstrates how to export the recovered drift and diffusion data as CSV files or Pandas data-frames.

3.1.6 Fitting non-polynomial functions (Experimental)

By default, PyDaddy fits polynomials for the drift and diffusion functions. However, this behaviour can be customized by providing a custom library of functions for the sparse regression procedure, this notebook demonstrates how to do this. (This is an experimental feature and not all functionality will work with non-polynomial functions).

3.1.7 Example analysis: Fish schooling

An example analysis of a fish schooling dataset (Jhawar et. al., Nature Physics, 2020) using PyDaddy.

3.1.8 Example analysis: Cell hopping

An example analysis of a confined cell migration dataset (Brückner et. al., Nature Physics, 2019) using PyDaddy.

3.2 Installation Guide

3.2.1 Prerequisites

You need to have a working Python 3 environment on your system. The recommended way to get a Python 3 installation is using Miniconda (minimal installation, very small download) or Anaconda (comes with many packages pre-installed, so a much larger ~3GB download: but a bit more beginner-friendly to install). See the respective pages for more information.

Anaconda: https://www.anaconda.com

Miniconda: https://docs.conda.io/en/latest/miniconda.html

Some documentation on Anaconda vs Miniconda: https://docs.conda.io/projects/conda/en/latest/user-guide/install/download.html#anaconda-or-miniconda

---

Note: If you are comfortable working with Conda environments, we recommend creating a new environment for PyDaddy, but this is not strictly necessary. For more information about environments, see: https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html)

If you are installing the development version (see below), also ensure that you have Git installed on your system. You can do this using the following command:

```
git --version
```

(If Git is already installed, this should show the version number; otherwise, it will show an error message saying git is not found.)

If Git is not installed, download and install it from https://git-scm.com

---

Note: On macOS/Linux, the following commands needs to be entered in the Terminal. On a Windows machine with Anaconda/Miniconda, these need to be entered into the 'Anaconda Terminal' app.

3.2.2 Installation using `pip`

Once Python 3 is set up, the latest release version can be installed using the following command:

```
pip install pydaddy
```

To install the latest development version of PyDaddy directly from the GitHub repo, use:

```
pip install git+https://github.com/tee-lab/PyDaddy.git
```

3.2.3 Installation using `conda`

The release version of PyDaddy can also be installed using conda as follows:

```
conda install -c tee-lab pydaddy
```

3.2.4 Verifying installation

To verify that everything is installed and working correctly, close and reopen the Terminal (Anaconda Terminal if you are using Windows) and try the following. First open an IPython terminal as follows:

```
ipython
```

Within IPython, type the following:

```
import pydaddy
```

If the installation was sucessful, the import command should work and throw no errors. Now type

```
pydaddy.__version__
```

This should print the current version number of PyDaddy.

Congratulations, you have now successfully installed PyDaddy!

3.3 Usage Guide

There are multiple ways to use PyDaddy, and this page gives an overview of all of them.

3.3.1 PyDaddy on Google Colab

The easiest way to get started with PyDaddy is using Google Colaboratory (also called Colab). Colab is a free Jupyter notebook environment, by Google, hosted entirely in the cloud. You can run PyDaddy on Google Colab notebooks, without having to install anything on your system. The tutorial notebooks have links to open them directly on Colab. (Note that you will need a Google login.)

To use PyDaddy from *any* Colab notebook, enter the following line to a cell and run it. This command sets up PyDaddy on the notebook's environment.

```
%pip install git+https://github.com/tee-lab/PyDaddy.git
```

Note: To upload your data files to the Colab notebook, click the 'Files' icon on the sidebar, and then click the 'Upload to session storage' icon.

Warning: All files and data will be lost when you disconnect from the notebook. Make sure you download and save any relevant analysis results.

3.3.2 One-line operation

Once PyDaddy is installed on your machine, it can be invoked from the command-line using the single-command mode. This mode runs all relevant analysis on a specified data file, and generates a single HTML report with all the analysis results.

Note: If you installed PyDaddy in a separate conda environment, activate that environment before continuing.

To use PyDaddy in this mode, use the following command:

```
pydaddy <file-name> --column_format xyt
```

Replace `<file-name>` with the name of the CSV file containing the data to be analyzed. The CSV file should contain the one or two data columns and one optional time-stamp column. The columns could be in any order; and the column order can be speficied using the `--column_format` option as shown above. (For example, if the time-stamp column comes first followed by the x and y columns, the column format should be `txy`.) If time-stamp column is not present, the sampling interval can be provided using the `-t` option.

For more details about other options and flags, use

```
pydaddy --help
```

Note: Ideally, the one-line functionality should be used only for a quick preliminary analysis. In particular, the results of the function fitting may not be optimal and may contain spurious terms. For best results, use PyDaddy within a notebook or script to fine-tune the estimation procedure (see the advanced function fitting tutorial).

3.3.3 Python Interface

Note: If you installed PyDaddy in a separate conda environment, activate that environment before continuing.

For full control over the estimation procedure, you can use PyDaddy through Python scripts or notebooks. To use PyDaddy in a Jupyter notebook, start a Jupyter notebook server using the following command:

```
jupyter notebook
```

Create a new notebook and import PyDaddy as follows.

```
import pydaddy
```

You should be able to use all features of PyDaddy in the notebook. See the tutorials or *package documentation* for more details on available functionality.

3.4 Usage Tips

- While working with 1-D data, if your data is a 1-D array, remember to wrap the array in a list while passing to PyDaddy, like so: `pydaddy.Characterize([x], ...`. Otherwise, PyDaddy will throw an error.
- PyDaddy expects uniformly sampled time-series. If your dataset is sampled with irregular time intervals, resample the time-series to a uniform sampling interval before using PyDaddy (a library like traces will be useful for this).
- There is a `simulate()` function (*pydaddy.daddy.Daddy.simulate()*) provided, that can generate simulated time series using the SDE estimated by PyDaddy. If you need to do advanced testing or diagnostics using simulated data, use this function.
- When necessary, you can 'hack' the `simulate()` (*pydaddy.daddy.Daddy.simulate()*) function to use custom drift and diffusion functions, not necessarily the results of the fits. To do this, assign appropriate functions to `ddsde.F` and `ddsde.G` (`ddsde.F1`, `ddsde.F2`, `ddsde.G11`, `ddsde.G12`, `ddsde.G22` for vector).
- The `fit()` (*pydaddy.daddy.Daddy.fit()*) function has an `alpha` parameter, which is a ridge regularization parameter. This is useful when the data is noisy or has outliers. If you think `fit()` is tending to overfit the data, non-zero for `alpha` and see if the fits improve (very high values, as high as 10e6 or 10e7 may be often required to see noticable effects). Be aware of the fact that large values of alpha has a side-effect of shrinking the estimated parameters.
- If `noise_diagnostics()` (*pydaddy.daddy.Daddy.noise_diagnostics()*) suggests that the noise autocorrelation is too high, a straightforward way around this problem is to subsample the data until the noise-correlation goes away. PyDaddy provides an easy way to do this: initialize `Characterize()` (*pydaddy.characterize.Characterize*) with parameters `Dt=T, dt=T` where `T` is the autocorrelation time (in number of time-steps) rounded to the nearest integer. However, note that using larger values of `T` can distort the estimated

drift and diffusion functions. Specifically, when the sampling time is too high, the estimated drift will be linear and the estimated diffusion will be quadratic (regardless the shape of the actual drift and diffusion functions).

- If the estimated drift is linear and the estimated diffusion is quadratic, the analysis results may not be reliable, and additional checks may be required: as mentioned above, these results can appear spuriously when the sampling interval is high[1].
- Sometimes, the `model_diagnostics()` (*pydaddy.daddy.Daddy.model_diagnostics()*) can incorrectly suggest that the estimated model is inconsistent. This often happens when the sampling time of the data is too large. This can due to errors in the SDE simulation rather than a model inconsistency. To tackle this, `model_diagnostics()` has an `oversample` parameter that can be used to specify an oversampling factor and simulate with an integration timestep smaller than the sampling interval. See the function documentation (*pydaddy.daddy.Daddy.model_diagnostics()*) for more details.

3.5 Sample Datasets

PyDaddy comes pre-packaged with several sample datasets. These can be loaded easily using the `load_sample_dataset()` function.

```
pydaddy.load_sample_dataset(<dataset-name>)
```

The following sample datasets are available:

- `fish-data-etroplus`: Group polarization data from a fish schooling experiment[1].
- `model-data-scalar-pairwise` and `model-data-scalar-ternary`: Scalar (1-D) simulated datasets generated from a stochastic Gillespie simulation, with pairwise and ternary interaction models respectively[1][2].
- `model-data-vector-pairwise` and `model-data-vector-ternary`: Vector (2-D) simulated datasets generated from a stochastic Gillespie simulation, with pairwise and ternary interaction models respectively[1][2].

The fish schooling dataset contains the time series of the group polarization vector $mathbf m$ (2-dimensional), for a group of 15 fish (emph{Etroplus suratensis}). The polarization time series is available at a uniform interval of 0.12 second. The dataset contains many missing data points[1].

The simulated datasets were generated using a continuous-time stochastic simulation algorithm, with pairwise and ternary interaction models respectively. Each simulated time series was resampled to a suitable uniform sampling interval[1][2]. Simulated datasets are provided for both 1-D and 2-D.

3.6 Package Documentation

3.6.1 pydaddy.Characterize

class pydaddy.characterize.**Characterize**(*data*, *t=1.0*, *Dt=1*, *dt=1*, *bins=None*, *inc=None*, *inc_x=None*, *inc_y=None*, *n_trials=1*, *show_summary=True*, ***kwargs*)

Bases: `object`

Intialize a PyDaddy object for further analysis.

[1] Riera, R., & Anteneodo, C. (2010). Validation of drift and diffusion coefficients from experimental data. Journal of Statistical Mechanics: Theory and Experiment, 2010(04), P04020.

[1] Jhawar, J., Morris, R. G., Amith-Kumar, U. R., Danny Raj, M., Rogers, T., Rajendran, H., & Guttal, V. (2020). Noise-induced schooling of fish. Nature Physics, 16(4), 488-493.

[2] Jhawar, J., & Guttal, V. (2020). Noise-induced effects in collective dynamics and inferring local interactions from data. Philosophical Transactions of the Royal Society B, 375(1807), 20190381.

Parameters

- **data** (*list*) – Time series data to be analysed. data = [x] for scalar data and data = [x1, x2] for vector where x, x1 and x2 are of numpy.array object type
- **t** (*float, array, optional (default=1.0)*) – t can be either a float representing the time-interval between observations, or a numpy array containing the time-stamps of the individual observations (Note: PyDaddy only supports uniformly spaced time-series, even when time-stamps are provided).
- **bins** (*int, optional (default=20)*) – Number of bins for computing bin-wise averages of drift and diffusion (Binwise averages are used only for visualization.)
- **show_summary** (*bool, optional (default=True)*) – If true, a summary text and summary figure will be shown.
- **Dt** (*int, optional (default=1)*) – Subsampling factor for drift computation. When provided, the time-series will be sub-sampled by this factor while computing drift.
- **dt** (*int, optional (default=1)*) – Subsampling factor for diffusion computation. When provided, the time-series will be sub-sampled by this factor while computing diffusion.
- **inc** (*float, optional (default=0.01)*) – For scalar data, instead of specifying *bins*, the widths (increments) of the bins can also be provided.
- **inc_x** (*float, optional (default=0.1)*) – For vector data, instead of specifying *bins*, the widths (increments) of the bins can also be provided. inc_x is the increment in the x-dimension.
- **inc_y** (*float, optional (default=0.1)*) – For vector data, instead of specifying *bins*, the widths (increments) of the bins can also be provided. inc_y is the increment in the y-dimension.
- **n_trials** (*int, optional (default=1)*) – Number of trials, concatenated timeseries of multiple trials is used.

Returns

output – Daddy object which can be used for further analysis and visualization. See `pyaddy.daddy.Daddy` for details.

Return type

pydaddy.daddy.Daddy

`pydaddy.characterize.load_sample_dataset(name)`

Load one of the sample datasets. For more details on the datasets, see *Sample Datasets*.

Available data sets:

```
'fish-data-etroplus'
'cell-data-cellhopping'
'model-data-scalar-pairwise'
'model-data-scalar-ternary'
'model-data-vector-pairwise'
'model-data-vector-ternary'
```

Parameters

name (*str*) – name of the data set

Returns

- **data** (*list*) – timeseries data

- **t** (*float, array*) – timescale

3.6.2 pydaddy.Daddy

class pydaddy.daddy.**Daddy**(*ddsde*, ***kwargs*)

Bases: `Preprocessing`, `Visualize`

An object of this type is returned by `pydaddy.daddy.Characterize`. This is the main workhorse class of PyDaddy, and contains functionality to compute drift and diffusion, visualize results, and perform diagnostic tests. See the individual method documentation for more details.

autocorrelation(*lags=1000*)

Show the autocorrelation plot of the data.

cross_diffusion(*slider_timescales=None*, *limits=None*, *polar=False*, ***plot_text*)

Show an interactive figure of the cross-diffusion function (only for vector data). The bin-wise averaged estimates of the drift will be shown. If a polynomial has already been fitted with fit() function, a line (scalar)/surface (vector) plot of the fitted function will also be shown.

Parameters

- **limits** (*tuple, (default=None)*) – If specified, sets the y-axis limits for the cross diffusion function. Useful to get a clearer view when there are outliers.
- **polar** (*bool, (default=False):*) – If True, plot the cross diffusion function only within a unit circle. Useful to get a better visualization in situations where |M| has an upper bound. (Used only in vector case).
- ****plot_text** (*dict:*) – To customize the captions, labels and layout of the plot, plot parameters can be passed as a dict. Available options are given below:

  For scalar analysis
  x_lable : x axis label

  y_label : y axis label

  For vector analysis
  title1 : first plot title

  x_label1 : first plot x label

  y_label1 : first plot y label

  z_label1 : first plot z label

  title2 : second plot title

  x_label2 : second plot x label

  y_label2 : seocnd plot y label

  z_label2 : second plot z label

diffusion(*slider_timescales=None*, *limits=None*, *polar=False*, ***plot_text*)

Show an interactive figure of the diffusion function. The bin-wise averaged estimates of the drift will be shown. If a polynomial has already been fitted with fit() function, a line (scalar)/surface (vector) plot of the fitted function will also be shown.

Parameters

- **limits** (*tuple, (default=None)*) – If specified, sets the y-axis limits for the diffusion function. Useful to get a clearer view when there are outliers.

- **polar** (*bool, (default=False):*) – If True, plot the diffusion function only within a unit circle. Useful to get a better visualization in situations where |M| has an upper bound. (Used only in vector case).
- ****plot_text** (*dict:*) – To customize the captions, labels and layout of the plot, plot parameters can be passed as a dict. Available options are given below:

  For scalar analysis
  x_lable : x axis label

  y_label : y axis label

  For vector analysis
  title1 : first plot title

  x_label1 : first plot x label

  y_label1 : first plot y label

  z_label1 : first plot z label

  title2 : second plot title

  x_label2 : second plot x label

  y_label2 : seocnd plot y label

  z_label2 : second plot z label

drift(*limits=None*, *polar=False*, *slider_timescales=None*, ***plot_text*)

Show an interactive figure of the drift function. The bin-wise averaged estimates of the drift will be shown. If a polynomial has already been fitted with fit() function, a line (scalar)/surface (vector) plot of the fitted function will also be shown.

Parameters

- **limits** (*tuple, (default=None)*) – If specified, sets the y-axis limits for the drift function. Useful to get a clearer view when there are outliers.
- **polar** (*bool, (default=False):*) – If True, plot the drift function only within a unit circle. Useful to get a better visualization in situations where |M| has an upper bound. (Used only in vector case).
- ****plot_text** (*dict:*) – To customize the captions, labels and layout of the plot, plot parameters can be passed as a dict. Available options are given below:

  For scalar analysis
  x_lable : x axis label

  y_label : y axis label

  For vector analysis
  title1 : first plot title

  x_label1 : first plot x label

  y_label1 : first plot y label

  z_label1 : first plot z label

  title2 : second plot title

  x_label2 : second plot x label

  y_label2 : seocnd plot y label

  z_label2 : second plot z label

export_data(*filename=None*, *raw=False*)

Returns a pandas dataframe containing the drift and diffusion values. Optionally, the data is also saved as a CSV file.

Parameters

- **filename** (*str, optional(default=None)*) – If provided, the data will be saved as a CSV at the given path. Else, a dataframe will be returned.
- **raw** (*bool, optional(default=False)*) – If True, the raw, the drift and diffusion will be returned as raw unbinned data. Otherwise (default), drift and diffusion as binwise-average Kramers-Moyal coefficients are returned.

Returns
DataFrame

Return type
Pandas dataframe containing the estimated drift and diffusion coefficients.

fit(*function_name*, *order=None*, *threshold=0.05*, *alpha=0*, *tune=False*, *thresholds=None*, *library=None*, *plot=False*)

Fit analytical expressions to drift/diffusion functions using sparse regression. By default, a polynomial with a specified maximum degree will be fitted. Alternatively, you can also provide a library of custom functions for fitting.

Parameters

- **function_name** (*str,*) – Name of the function to fit. Can be 'F' or 'G' for scalar; 'F1', 'F2', 'G11', 'G22', 'G12', 'G21' for vector
- **order** (*int,*) – Order (maximum degree) of the polynomial to fit.
- **threshold** (*float, (default=0.05)*) – Sparsification threshold
- **tune** (*bool, (default=False)*) – If True, the sparsification threshold will be automatically set using cross-validation.
- **alpha** (*float, (default=0.0)*) – Optional regularization term for ridge regression. Useful when data is too noisy, but has a side effect of shrinking the estimated coefficients when set to high values.
- **thresholds** (*list, (default=None)*) – With tune=True, a list of thresholds over which to search for can optionally be provided. If not present, this will be chosen automatically as the range between the minimum and maximum coefficients in the initial fit.
- **library** (*list, (default=None)*) – A custom library of non-polynomial functions can optionally be provided. If provided, the functions will be fitted as a sparse linear combination of the terms in the library.

Returns
res

Return type
fitters.Poly1D or fitters.Poly2D object, representing the fitted polynomial.

model_diagnostics(*oversample=1*)

Perform model self-consistency diagnostics. Generates a simulated time series with the same length and sampling interval as the original time series, and re-estimates the drift and diffusion from the simulated time series. The re-estimated drift and diffusion should match the original estimates.

Parameters

- **oversample** (*int, (default=1)*) – Factor by which to oversample while simulating the SDE. If provided, the SDE will be simulated at *t_int / oversample* and then subsampled to *t_int*. This is useful when *t_int* is large enough to cause large errors in the SDE simulation.
- **plotted** (*The following are*) –
  - Histogram of the original time series overlaid with that of the simulated time series.
  - Drift and diffusion of the original time series overlaid with that of the simulated time series.

noise_diagnostics(*loc=None*)

Perform diagostics on the noise-residual, to ensure that all assumptions for SDE estimation are met. Generates a plot with:

- Distribution (1D or 2D histogram) of the residuals, and their QQ-plots against a theoretically expected Gaussian. The residual distribution is expected to be a Gaussian.
- Autocorrelation of the residuals. The autocorrelation time should be close to 0.
- Plot of the 2nd versus 4th jump moments. This plot should be a straight line. (Only for scalar data.)

parameters()

Get all given and assumed parameters used for the analysis

Returns
params – all parameters given and assumed used for analysis

Return type
dict, json

simulate(*t_int*, *timepoints*, *x0=None*)

Generate simulated time-series with the fitted SDE model.

Generates a simulated timeseries, with specified sampling time and duration, based on the SDE model discovered by PyDaddy. The drift and diffusion functions should be fit using fit() function before using simulate().

Parameters

- **t_int** (*float*) – Sampling time for the simulated time-series
- **timepoints** (*int*) – Number of time-points to simulate
- **x0** (*float (scalar) or list of two floats (vector), (default=None)*) – Initial condition. If no value is passed, 0 ([0, 0] for vector) is taken as the initial condition.

Returns
x

Return type
Simulated timeseries with *timepoints* timepoints.

summary(*start=0*, *end=1000*, *kde=True*, *tick_size=12*, *title_size=15*, *label_size=15*, *label_pad=8*, *n_ticks=3*, *ret_fig=False*, ***plot_text*)

Print summary of data and show summary plots chart. (This is the same summary plot produced by Characterize().)

Parameters

- **start** (*int, (default=0)*) – starting index, begin plotting timeseries from this point
- **end** (*int, default=1000*) – end point, plots timeseries till this index

- **kde** (*bool, (default=False)*) – if True, plot kde for histograms
- **title_size** (*int, (default=15)*) – title font size
- **tick_size** (*int, (default=12)*) – axis tick size
- **label_size** (*int, (default=15)*) – label font size
- **label_pad** (*int, (default=8)*) – axis label padding
- **n_ticks** (*int, (default=3)*) – number of axis ticks
- **ret_fig** (*bool, (default=True)*) – if True return figure object
- ****plot_text** – plots' title and axis texts

  For scalar analysis summary plot:

  timeseries_title : title of timeseries plot

  timeseries_xlabel : x label of timeseries

  timeseries_ylabel : y label of timeseries

  drift_title : drift plot title

  drift_xlabel : drift plot x label

  drift_ylabel : drift plot ylabel

  diffusion_title : diffusion plot title

  diffusion_xlabel : diffusion plot x label

  diffusion_ylabel : diffusion plot y label

  For vector analysis summary plot:

  timeseries1_title : first timeseries plot title

  timeseries1_ylabel : first timeseries plot ylabel

  timeseries1_xlabel : first timeseries plot xlabel

  timeseries1_legend1 : first timeseries (Mx) legend label

  timeseries1_legend2 : first timeseries (My) legend label

  timeseries2_title : second timeseries plot title

  timeseries2_xlabel : second timeseries plot x label

  timeseries2_ylabel : second timeseries plot y label

  2dhist1_title : Mx 2d histogram title

  2dhist1_xlabel : Mx 2d histogram x label

  2dhist1_ylabel : Mx 2d histogram y label

  2dhist2_title : My 2d histogram title

  2dhist2_xlabel : My 2d histogram x label

  2dhist2_ylabel : My 2d histogram y label

  2dhist3_title : M 3d histogram title

  2dhist3_xlabel : M 2d histogram x label

  2dhist3_ylabel : M 2d histogram y label

3dhist_title : 3d histogram title

3dhist_xlabel : 3d histogram x label

3dhist_ylabel : 3d histogram y label

3dhist_zlabel : 3d histogram z label

driftx_title : drift x plot title

driftx_xlabel : drift x plot x label

driftx_ylabel : drift x plot y label

driftx_zlabel : drift x plot z label

drifty_title : drift y plot title

drifty_xlabel : drift y plot x label

drifty_ylabel : drift y plot y label

drifty_zlabel : drift y plot z label

diffusionx_title : diffusion x plot title

diffusionx_xlabel : diffusion x plot x label

diffusionx_ylabel : diffusion x plot y label

diffusionx_zlabel : diffusion x plot z label

diffusiony_title : diffusion y plot title

diffusiony_xlabel : diffusion y plot x label

diffusiony_ylabel : diffusion y plot y label

diffusiony_zlabel : diffusion y plot z label

Return type
None, or figure

Raises
ValueError – If start is greater than end

3.7 Citing PyDaddy

If you are using this package in your research, please cite the repository and the associated paper as follows:

Nabeel, A., Karichannavar, A., Palathingal, S., Jhawar, J., Danny Raj, M., & Guttal, V. (2022). PyDaddy: A Python Package for Discovering SDEs from Time Series Data (Version 0.1.5) [Computer software]. https://github.com/tee-lab/PyDaddy

Nabeel, A., Karichannavar, A., Palathingal, S., Jhawar, J., Danny Raj, M., & Guttal, V. (2022). PyDaddy: A Python package for discovering stochastic dynamical equations from timeseries data. arXiv preprint arXiv:2205.02645.

PYTHON MODULE INDEX

p

INDEX