pydaddy

Release 1.0.0

Ashwin Karichannavar

Apr 28, 2022

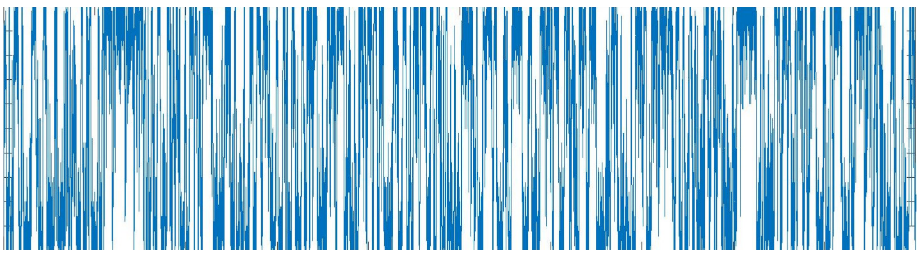
CONTENTS

pydaddy : a package to derive governing stocastic differential equations from timeseries data.

CHAPTER

ONE

PYDADDY



1.1 pydaddy

A package to derive an SDE equation form the data.

pydaddy is a python package implementing a data driven SDE method. pydaddy provides an interface which takes time series data as input, runs the analysis and returns an output object through which data and analysed results can be systematically visualized and saved.

1.2 How it works

Suppose m(t) is a SDE time series data.

The the package calculates the deterministic (drift) and stochastic (diffusion) component of dm/dt.

This data driven SDE method is based on the assumption that the noise in the time series is uncorrelated and Gaussian in nature, with zero mean and unit variance.

pydaddy extracts the noise form the data and checks if it holds true to its assumptions.

1.3 Features

- Simple one line execution of analysis.
- Produces intuitive visuals of data and obtained result.
- Supports time series data with both scalar and vector order parameters.
- Generates interactive, drift and diffusion sliders for user defined timescale range or list.
- Diagnostics help understand change in order of drift and diffusion with varying timescale.

1.4 Requirements

Python:

- python version >=3.5 , <=3.8

Packages dependencies:

- statsmodels, version 0.11.1
- matplotlib, version 3.2.2
- scipy, version 1.5.2
- numpy, version 1.19.1
- seaborn, version 0.10.1
- tqdm, version 4.48.2
- plotly, version 4.14.3

1.5 Installation

pydaddy is available both on pypi and anaconda cloud, which offers two recommended methods of installations, using pip python package manager or anaconda package manager.

Note: To run the example notebooks on your system after installing the package, please download the notebook files to your PC from the github repo.

Alternately, its recommended to simply clone or download git the repository.

1.5.1 Installing via conda

Conda is a package manager that handles sourcing of all dependencies in a relatively straight-forward, cross-platform manner. pydaddy is available on the `tee-lab` channel.

Important: Make sure you have anaconda or miniconda installed and have activated the conda default (base) environment using `conda activate base` , before proceeding.

Install pydaddy package with all its dependencies using conda simply requires executing the command

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

To install pydaddy in a clean virtual environment use

```
conda create --name MY_ENV_NAME -c tee-lab pydaddy
```

Replace `MY_ENV_NAME` with your desired name for environment.

You can now access pydaddy package by activating the newly created environment, `conda activate MY_ENV_NAME`

1.5.2 Installing via pip

To install using pip package manager, run:

```
python -m pip install pydaddy
```

1.5.3 Manual installation

Installing Latest unreleased version

Without git, you will need to download the zip-file of the code, extract it and follow the above instructions.

Click here *to download source code zip file.*

This method is not recommended unless you experience problems with conda or pip. To install using setuptools, download the source code manually and run `python setup.py install` from the terminal.

This will install the package in your current environment (if you are working in any environment).

If you have git installed, you can clone the repo and install using the following commands.

```
$ git clone https://github.com/tee-lab/pydaddy.git
$ cd pydaddy
$ python setup.py install
```

Important: Without git, you will need to download the zip-file of the code, extract it and follow the above instructions.

Click here *to download source code zip file.*

1.6 Usage

The time series data is given as input to the `Characterize` method along with all other optional parameters.

Show `pydaddy.Characterize` documentation

Characterize

`class pydaddy.Characterize`(*data*, *t=1.0*, *Dt=1*, *dt=1*, *bins=None*, *inc=None*, *inc_x=None*, *inc_y=None*, *slider_timescales=None*, *n_trials=1*, *show_summary=True*, *drift_threshold=None*, *diff_threshold=None*, *drift_degree=5*, *diff_degree=5*, *drift_alpha=0*, *diff_alpha=0*, *fit_functions=False*, ***kwargs*)

Bases: `object`

Analyse a time series data and get drift and diffusion plots.

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)*) – float if its time increment between observation

  numpy.array if time stamp of time series
- **Dt** (*int,'auto', optional(default='auto')*) – time scale for drift

  if ‘auto’ time scale is decided based of drift order.
- **dt** (*int, optional(default=1)*) – time scale for difusion
- **inc** (*float, optional(default=0.01)*) – increment in order parameter for scalar data
- **inc_x** (*float, optional(default=0.1)*) – increment in order parameter for vector data x1
- **inc_y** (*float, optional(default=0.1)*) – increment in order parameter for vector data x2
- **fft** (*bool, optional(default=True)*) – if true use fft method to calculate autocorrelation else, use standard method
- **slider_timescales** (*list, optional(default=None)*) – List of timescale values to include in slider.
- **n_trials** (*int, optional(default=1)*) – Number of trials, concatenated timeseries of multiple trials is used.
- **show_summary** (*bool, optional(default=True)*) – print data summary and show summary chart.
- ****kwargs** – all the parameters for inherited methods.

Returns **output** – object to access the analysed data, parameters, plots and save them.

Return type *pydaddy.output.Output*

1.6.1 Key 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 types.

t: array or float float if its time increment between observation numpy.array if time stamp of time series

See doc strings or code documentation for more information.

1.6.2 Example using sample data set

See *Data set description* for more information about the datasets.

```
import pydaddy
#load data
data, t = pydaddy.load_sample_dataset('model-data-vector-ternary')
# Analyse
ddsde = pydaddy.Characterize(data,t)

# Show drift slider plot
ddsde.drift()
# Show diffuision slider plot
ddsde.diffusion()
# Show timeseries plot
ddsde.timeseries()
# Show histograms
ddsde.histograms()
# Show all inputed, calculated and assumed parameters of the analysis
ddsde.parameters()
# Export data to disk
ddsde.export_data()
```

`Characterize` returns an output object in which all analysed results are stored. Results can be visualised or stored by calling appropriate functions:

Show `pydaddy.output.output` documentation

---

pydaddy.output.output

`pydaddy.output`

alias of <module 'pydaddy.output' from '/home/docs/checkouts/readthedocs.org/user_builds/pyddsde/checkouts/latest/pydadd

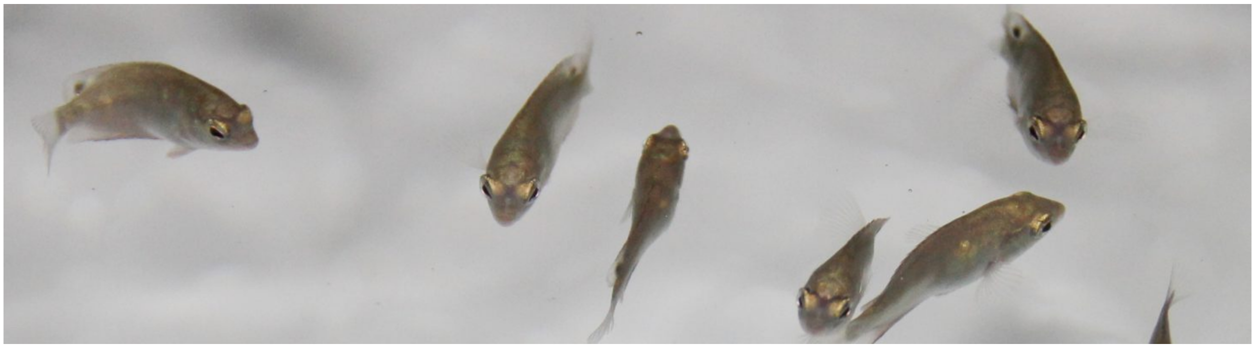
---

- `summary()`: show summary
- `drift()` : drift slider plot
- `diffusion()` : diffusion slider plot
- `timeseries()`: time series plot
- `histograms()` : histogram plots
- `noise_characterstics()`: noise characteristics plots
- `visualise(timescale)`: drift and diffusion plots for a timescale
- `diagnostics()`: diagnostics plots

- `data(timescale)`: get drift and diffusion data for a timescale
- `export_data()`: Save data as csv files and mat files
- `plot_data(data)`: plot data on a 3d axis

For more examples see this notebook.

1.7 Motivation



This project is motivated by the study of group behaviour dynamics of animals, especially schooling fish. The sample data distributed along with this package is from experiments conducted by TEElab, IISc.

1.8 Data set description

pydaddy has six data set included along with the package which can be loaded using `load_sample_dataset(dataset_name)` function

Show `pydaddy.load_sample_dataset` documentation

---

pydaddy.load_sample_dataset

`class pydaddy.load_sample_dataset(`*name*`)`

Bases:

Load sample data set provided.

Available data sets:

'fish-data-etroplus'

'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

1.8.1 Experiment data (from experimentation or from observations)

fish-data-ectropus: A data from experiment conducted with a group of 30 fish, in which the group polarity in x and y directions are recorded every 0.12 seconds.

Source

- The fish data is a part of the work done in the *Noise-Induced Schooling of Fish*

1.8.2 Simulation data

A synthetic data set obtained from the simulation of fish interactions.

model-data-vector-pairwise : Pairwise interaction of fish simulated in two dimension.

model-data-vector-ternary: Ternary interaction of fish simulated in two dimension.

model-data-scalar-pairwise: Pairwise interaction of fish simulated in single dimension.

model-data-scalar-ternary: Ternary interaction of fish simulated in single dimension.

Source

- The simulation method is inspired from the work done in *Noise-induced Effects in Collective Dynamics and Inferring Local Interactions from Data*

1.9 Glossary

drift First jump moment

diffusion second jump moment

1.10 Acknowledgement

1.11 Licence

1.12 Reference

[1] Noise-induced Effects in Collective Dynamics and Inferring Local Interactions from Data Preprint Github

[2] Noise-Induced Schooling of Fish Preprint Github

CHAPTER

TWO

CODE EXAMPLES

2.1 Code Examples

Example Codes will be here

CHAPTER

THREE

INSTALLATION GUIDE

3.1 pyddsde Installation Guide

3.1.1 Anaconda

This guide explains creating and installing pyddsde package.

If you don't have Anaconda installed, you can do so from Anaconda Website

Step 1 : Clone the git repo

Open the terminal in your prefered directory and execute the below command to clone the repo

```
git clone https://github.com/tee-lab/pyFish.git
```

After cloning the terminal should look like this

Step 2 : Change the directory to pyddsde

```
cd pyddsde
```

Typing `ls` should show the following content

Step 3 : Create python environment

```
conda env create -f environment.yml
```

Now, an environment named `pyddsde` should be created

Step 4 : Activate pyddsde environment

```
conda activate pyddsde
```

The (pyddsde) should appear in the terminal.

Step 5 : Install pyddsde

```
python -m pip install .
```

If you see a similar output at the end then the package is successfully installed

You can run the notebook files using jupyter notebook (or jupyter lab)

```
jupyter notebook
```

This should open the notebook application in the browser, click on `notebooks` folder and open the .ipynb notebook file.

After opening the file, click on the cell and press `Shift+Enter` to execute that cell and move to the next.

3.1.2 pip

CHAPTER

FOUR

CODE DOCUMENTATION

4.1 Modules

4.1.1 Key Modules

Characterize

class pydaddy.**Characterize**(*data*, *t=1.0*, *Dt=1*, *dt=1*, *bins=None*, *inc=None*, *inc_x=None*, *inc_y=None*, *slider_timescales=None*, *n_trials=1*, *show_summary=True*, *drift_threshold=None*, *diff_threshold=None*, *drift_degree=5*, *diff_degree=5*, *drift_alpha=0*, *diff_alpha=0*, *fit_functions=False*, ***kwargs*)

Bases: `object`

Analyse a time series data and get drift and diffusion plots.

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)*) – float if its time increment between observation

  numpy.array if time stamp of time series
- **Dt** (*int,'auto', optional(default='auto')*) – time scale for drift

  if 'auto' time scale is decided based of drift order.
- **dt** (*int, optional(default=1)*) – time scale for difusion
- **inc** (*float, optional(default=0.01)*) – increment in order parameter for scalar data
- **inc_x** (*float, optional(default=0.1)*) – increment in order parameter for vector data x1
- **inc_y** (*float, optional(default=0.1)*) – increment in order parameter for vector data x2
- **fft** (*bool, optional(default=True)*) – if true use fft method to calculate autocorrelation else, use standard method
- **slider_timescales** (*list, optional(default=None)*) – List of timescale values to include in slider.
- **n_trials** (*int, optional(default=1)*) – Number of trials, concatenated timeseries of multiple trials is used.

- **show_summary** (*bool, optional(default=True)*) – print data summary and show summary chart.
- ****kwargs** – all the parameters for inherited methods.

Returns **output** – object to access the analysed data, parameters, plots and save them.

Return type *pydaddy.output.Output*

load_sample_dataset

class pydaddy.**load_sample_dataset**(*name*)

Bases:

Load sample data set provided.

Available data sets:

'fish-data-etroplus'

'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

output

class pydaddy.output.**Output**(*ddsde*, ***kwargs*)

Bases: *pydaddy.preprocessing.Preprocessing*, *pydaddy.visualize.Visualize*

Class to plot and save data and parameters

property A1

property A2

property B11

property B12

property B21

property B22

property F

property G

property _data_avgdiff

property `_data_avgdiffX`

property `_data_avgdiffXY`

property `_data_avgdiffY`

property `_data_avgdiffYX`

property `_data_avgdrift`

property `_data_avgdriftX`

property `_data_avgdriftY`

property `_data_diff_ebar`

property `_data_drift_ebar`

`_print_function_diagnostics`(*f*, *x*, *y*, *name*, *symbol*)

`_print_function_diagnostics_2d`(*f*, *x*, *y*, *z*, *name*, *symbol*)

`_update_slider_data`(*slider_timescales*)

`autocorrelation`()

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

Display diffusion cross correlation slider figure

Parameters `polynomial_order` (*None or int, default=None*) – order of polynomial to fit, if None, no fitting is done.

Returns **opens diffusion slider**

Return type None

`data`(*drift_time_scale=None*, *diff_time_scale=None*)

Get the drift, diffusion and order parameter data for any timescale the analysis is done.

Parameters

- `drift_time_scale` (*int, optional(default=None)*) – time-scale of drift data, if None, returns data analysed for given dt
- `diff_time_scale` (*int, optional(default=None)*) – time-scale of diffusion data, if None, returns data analysed for given delta_t

Returns

- if vector, [avgdriftX, avgdriftY, avgdiffX, avgdiffY, op_x, op_y]
- else, [avgdrift, avgdiff, op]

Return type list

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

Display diffusion slider figure

Parameters `polynomial_order` (*None or int, default=None*) – order of polynomial to fit, if None, no fitting is done.

Returns **opens diffusion slider**

Return type None

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

Display drift slider figure

Parameters

- **polynomial_order** (*None or int, default=None*) – order of polynomial to fit, if None, no fitting is done.
- ****plot_text** – plots' axis and text label

  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

Returns **opens drift slider**

Return type None

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

Returns a pandas dataframe containing the drift and diffusion values. :param filename: If provided, the data will be saved as a CSV at the given path. Else, a dataframe will be returned. :type filename: str, optional (default=None). :param raw: 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 **df**

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

`fit_diagnostics`()

`histogram`(*kde=False*, *heatmap=False*, *dpi=150*, *title_size=14*, *label_size=15*, *tick_size=12*, *label_pad=8*, ***plot_text*)

Show histogram polt chart

Parameters

- **kde** (*bool, (default=False)*) – If True, plots kde for histograms
- **dpi** (*int, (defautl=150)*) – figure resolution
- **title_size** (*int, (default=14)*) – title font size
- **label_size** (*int, (default=15)*) – axis label font size
- **tick_size** (*int, (default=12)*) – axis ticks font size
- **label_pad** (*int, (default=8)*) – axis label padding

- **plot_text – plots' axis and title text

  For scalar analysis histograms: hist_title : title

  hist_xlabel : x label

  hist_ylabel : y label

  For vector analysis histograms: hist1_title : first histogram title

  hist1_xlabel : first histogram x label

  hist1_ylabel : first histogram y label

  hist2_title : second histogram title

  hist2_xlabel : second histogram x label

  hist2_ylabel : second histogram y label

  hist3_title : third histogram title

  hist3_xlabel : third histogram x label

  hist3_ylabel : third histogram y label

  hist4_title : fourth (3d) histogram title hist4_xlabel : fourth (3d) histogram x label hist4_ylabel : fourth (3d) histogram y label hist4_zlabel : fourth (3d) histogram z label

Returns **histogam chart**

Return type matplotlib.pyplot.figure

`noise_diagnostics`(*loc=None*)

`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

`plot_data`(*data_in*, *ax=None*, *clear=False*, *title=None*, *x_label='x'*, *y_label='y'*, *z_label='z'*, *tick_size=12*, *title_size=16*, *label_size=14*, *label_pad=12*, *legend_label=None*, *dpi=150*)

Plot and visualize vector drift or diffusion data of a 3d axis

Can be used plot multiple data on the same figure and compare by passing the axis of figure.

Parameters

- **data_in** (*numpy.array*) – vector drift or diffusion data to plot
- **ax** (*figure axis, (default=None)*) – Ia ax is None, a new axis will be created and data will be plotted on it.
- **clear** (*bool, (default=False)*) – if True, clear the figure.
- **title** (*str, (default=None)*) – title of the figure
- **x_label** (*str, (default='x')*) – x-axis label
- **y_label** (*str, (default='y')*) – y-axis label
- **z_label** (*str, (default='z')*) – z-axis label
- **tick_size** (*int, (default=12)*) – axis ticks font size
- **title_size** (*int, (default=16)*) – title font size

- **label_size** (*int, (default=14)*) – axis label font size
- **label_pad** (*int, (default=12)*) – axis label padding
- **legend_label** (*str, (default=None)*) – data legend label
- **dpi** (*int, (default=150)*) – figure resolution

Returns

- **ax** (*3d figure axis*) – axis of the 3d figure.
- **fig** (*matplotlib figure*) – returns figure only if the input ax is None.

`release()`

Clears the memory, recommended to be used while analysing multiple data files in loop.

Return type None

`simulate(sigma=4, dt=None, T=None, **functions)`

Simulate SDE

Takes drift and diffusion functions as input and simuates the SDE using the analysis parameters.

The drift and diffusion functions given must be callable type functions.

For scalar F and G (drift and diffusion) must take one input and return a number

Parameters

- **sigma** (*float*) – magnitude of the noise eta(t)
- ****functions** – drift and diffusion callable functions

  For scalar analysis F : drift function

  G : diffusion dunction

  For vector analysis A1 : drift X

  A2 : drift Y

  B11 : diffusion X

  B22 : diffusion Y

  B12 : diffusion XY

  B21 : diffusion YX

Returns

simulated timeseries – [M] if scalar

[Mx, My] is vector

Return type list

Examples

For scalar analysis def drift_function(x):

return 0.125 * x

def diffusion_function(x): return -(x**2 + 1)

simulated_data = ddsde.simulate(F=drift_function, G=diffusion_function)

For vector analysis def drift_x(x, y):

return x*x + y*y * x*y**2

def dirft_y(x, y): return x*y

def diffusion_x(x,y): return x**2 + x*y

def diffusion_y(x,y): return y**2 + x*y

def diffusion_xy(x,y): return 0

def diffusion_yx(x,y): rerutn 0

simulated_data = ddsde.simulate(A1=drift_x, A2=drift_y, B11=diffusion_x, B22=diffusion_y, B12=diffusion_xy. B21=diffusion_yx)

`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

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

`timeseries`(*start=0*, *end=1000*, *n_ticks=3*, *dpi=150*, *tick_size=12*, *title_size=14*, *label_size=14*, *label_pad=0*, ***plot_text*)

Show plot of input data

Parameters

- **start** (*int, (default=0)*) – starting index, begin plotting timeseries from this point
- **end** (*int, default=1000*) – end point, plots timeseries till this index
- **n_ticks** (*int, (default=3)*) – number of axis ticks
- **dpi** (*int, (default=150)*) – dpi of the figure
- **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
- ****plot_text** – plots' title and axis texts

  For scalar analysis plot: timeseries_title : title

  timeseries_xlabel : x label

  timeseries_ylabel : y label

  For vector analysis plot: timeseries1_title : first timeseries plot title

  timeseries1_xlabel : first timeseries plot x label

  timeseries1_ylabel : first timeseries plot y lable

  timeseries2_title : second timeseries plot title

  timeseries2_xlabel : second timeseries plot x label

  timeseries2_ylabel : second timeseries plot y label

  timeseries3_title : third timeseries plot title

  timeseries3_xlabel : third timeseries plot x label

  timeseries3_ylabel : third timeseries plot y label

Returns **time series plot**

Return type matplotlib.pyplot.figure

Raises `ValueError` – If start is greater than end

`visualize`(*drift_time_scale=None*, *diff_time_scale=None*)

Display drift and diffusion plots for a time scale.

Parameters **time_scale** (*int, optional(default=None)*) – timescale for which drift and diffusion plots need to be shown. If None, displays the plots for inputed timescale.

Returns **displays plots**

Return type None

4.1.2 Additional Modules

SDE

class pydaddy.sde.**SDE**(***kwargs*)

Bases: object

A class to form a basic SDE from data

Parameters

- **X** (*array_like*) – time series data
- **t_int** (*float*) – time step in time series
- **Dt** (*int*) – analysis time step
- **inc** (*float*) – max increment for binning thae data

Returns

- **diff** (*array_like*) – diffusion in time series
- **drift** (*array_like*) – drift in time series
- **avgdiff** (*array_like*) – avarage diffusion
- **avgdrift** (*array_like*) – average drift
- *meta private:*

_diffusion(*X*, *t_int*, *dt=1*)

Get Diffusion coefficient vector of data

Parameters

- **X** (*array_like*) – time series data
- **t_int** (*float*) – time step in time series
- **dt** (*int*) – diffusion calculation timescale

Returns **diff** – Diffusion

Return type array.

_diffusion_from_residual(*X*, *F*, *t_int*, *dt=1*)

Get diffusion using residuals about drift function.

Parameters

- **(np.array)** (*X*) –
- **(float)** (*t_int*) –
- **(Callable)** (*F*) –

_diffusion_x_from_residual(*x*, *y*, *A1*, *t_int*, *dt*)

_diffusion_xy(*x*, *y*, *t_int*, *dt*)

Get cross-correlation coefficients between x and y arrays.

Parameters

- **x** (*numpy.array*) – x data
- **y** (*numpy.array*) – y data
- **t_int** (*float*) – time difference betwen consecutive observations
- **dt** (*diffusion calculation timescale*) –

Returns **diffusion_xy** – cross-correlation coefficients between x and y data

Return type numpy.array

_diffusion_xy_from_residual(*x*, *y*, *A1*, *A2*, *t_int*, *dt*)

_diffusion_y_from_residual(*x*, *y*, *A2*, *t_int*, *dt*)

_diffusion_yx(*x*, *y*, *t_int*, *dt*)

Get cross-correlation coefficients between x and y arrays.

Parameters

- **x** (*numpy.array*) – x data
- **y** (*numpy.array*) – y data
- **t_int** (*float*) – time difference betwen consecutive observations
- **dt** (*diffusion calculation timescale*) –

Returns **diffusion_xy** – cross-correlation coefficients between y and x data

Return type numpy.array

_drift(*X*, *t_int*, *Dt*)

Get Drift coeffecient vector of data.

Parameters

- **X** (*array_like*) – Time Series data
- **t_int** (*float*) – time difference betwen consecutive observations
- **Dt** (*float*) – drift calculation timescale

Returns **diff** – Diffusion in time series

Return type array_like

Notes

Drift is calculated as follows

$$drift = \frac{x(i + Dt) - x(i)}{tint * Dt}$$

_drift_and_diffusion(*X*, *t_int*, *Dt*, *dt*, *inc*, *drift_threshold*, *drift_degree*, *drift_alpha*, *diff_threshold*, *diff_degree*, *diff_alpha*, *fast_mode*)

Get drift and diffusion coefficients for a given timeseries data

Parameters

- **X** (*numpy.array*) – time series data
- **t_int** (*float*) – time difference betwen consecutive observations
- **Dt** (*int*) – timescale to calculate drift
- **dt** (*int*) – timescale to claculate diffusion
- **inc** (*float*) – step increments in order parameter
- **drift_threshold** (*float or None*) – threshold to use for fitting drift function. If None, automatic model selection will be used.
- **diff_threshold** (*float or None*) – threshold to use for fitting diffusion function. If None, automatic model selection will be used.

Returns

- —
- **diff** (*array*) – diffusion of the data
- **drift** (*array.*) – drift, of the data
- **avgdiff** (*array*) – average diffusion
- **avgdrift** (*array*) – average drift
- **op** (*array*) – order parameter

_isValidRange(*r*)

Checks if the specified range of order parameter is valid range

Parameters **r** (*tuple, list*) – range of order parameter

Returns True if valid, False if not.

Return type bool

_km_coefficient(*order*, *X*, *t_int*)

_order_parameter(*X*, *inc*, *r*)

Get order parameter array for a given range and increments

If range is None or not valid, order parameter array will be generated considering maxmium and mimimum limits of the data as the range

Parameters

- **X** (*numpy.array*) – data
- **inc** (*float*) – step increments in order parameter
- **r** (*tuple, list*) – range of the order parameter

Returns first element will be the order parameter array second element is the rangen used

Return type tuple

_residual(*X*, *t_int*, *Dt*, *dt=1*)

Get the residual.

_vector_drift_diff(*x*, *y*, *inc_x*, *inc_y*, *t_int*, *Dt*, *dt*, *drift_threshold*, *drift_degree*, *drift_alpha*, *diff_threshold*, *diff_degree*, *diff_alpha*, *fast_mode*)

Get average binned drift and diffusion coefficients for given x and y data

Parameters

- **x** (*array_like*) – timeseries x data
- **y** (*array_like*) – timesereis y data
- **inc_x** (*float*) – step increment of order parameter for x
- **inc_y** (*float*) – step increment of order parameter for y
- **Dt** (*int*) – timescale to calculate drift
- **dt** (*int*) – timescale to calculate diffusion

Returns [avgdriftX, avgdriftY, avgdiffX, avgdiffY, avgdiffXY, op_x, op_y]

Return type list

metrics

class pydaddy.metrics.**Metrics**(***kwargs*)

Bases: object

Helper/utility module

_R2(*data*, *op*, *poly*, *k*, *adj=False*)

R-square value between the predicted and expected values

Parameters

- **data** (*array*) – depended variable values, expected values, data
- **op** (*array*) – independent variable values
- **poly** (*numpy.poly1d*) – numpy polynomial fitted object
- **k** (*int*) – degree of the polynomial poly
- **adj** (*bool*) – if True, use R2-adjusted method instead of R2

Returns **R2** – R2 or R2-adjusted depending upon 'adj' value

Return type float

_R2_adj(*data*, *op*, *poly*, *k*)

Get R-squared adjusted parameter between data and fitted polynomial

Parameters

- **data** (*array*) – depended variable values, expected values, data
- **op** (*array*) – independent variable for which the data is defined
- **poly** (*numpy.poly1d*) – numpy polynomial fitted object
- **k** (*int*) – degree of polynomial

Returns **R2-adjusted** – R2 adjusted parameter between data and fitted polynomial

Return type folat

`_closest_time_scale`(*time_scale*, *slider*)

Gives closest matching time scale avaiable from the slider keys.

`_combined_data_dict`()

Get all drift and diffusion data in dictionary format.

`_csv_header`(*prefix*, *file_name*)

Generate headers for CSV file.

`_divergence`(*a*, *b*)

Get the divergence between two timeseries data, the divergence returned here is defined as follows: divergence = 0.5*(KL_divergence(p,q) + KL_divergence(q,p))

The probablity density of a and b input timeseries is calculated before finding the divergence.

Parameters

- **a** (*array*) – observed timeseries data
- **b** (*array*) – simulated timeseries data

Returns **divergence**

Return type float

`_fit_plane`(*x*, *y*, *z*, *order=2*)

Fits n-th order plane to data in the form z = f(x,y) where f(x,y) the best fit equation of plane for the data computed using least square method.

Parameters

- **x** (*2D array*) – order parameter x
- **y** (*2D array*) – order parameter y
- **z** (*2D array*) – derrived drift or diffusion data
- **order** (*int*) – order of the 2D plane to fit

Returns A callable object takes in x and y as inputs and returns z = f(x,y), where f(x,y) is the fitted function of the plane.

Return type *pydaddy.metrics.Plane*

`_fit_poly`(*x*, *y*, *deg*)

Fits polynomial of degree *deg*

Parameters

- **x** (*array*) – independent variable
- **y** (*array*) – depended variable
- **deg** (*int*) – degree of the polynomial

Returns

- **poly** (*numpy.poly1d*) – polynomial object
- **x** (*array*) – values of x for where y in defined

Notes

The nan values in the input x and y (if any) will be ignored.

_fit_poly_sparse(*x*, *y*, *deg*, *threshold=0.05*, *alpha=0*, *weights=None*)

Fit a polynomial using sparse regression using STLSQ (Sequentially thresholded least-squares) :param x: (np.array) Independent and dependent variables :param y: (np.array) Independent and dependent variables :param deg: (int) Maximum degree of the polynomial :param threshold: (float) Threshold for sparse fit.

_get_data_from_slider(*drift_time_scale=None*, *diff_time_scale=None*)

Get drift and diffusion data from slider data dictionary, if key not valid, returns the data corresponding to closest matching one.

_get_data_range(*x*)

Get range of the values in x, (min(x), max(x)), rounded to 3 decimal places.

_get_num_points(*drift_time_scale*, *diff_time_scale*)

_get_stacked_data()

Get a dictionary of all (op_x, op_y, driftX, driftY, diffX, diffY) slider data stacked into numpy arrays.

_interpolate_missing(*y*, *copy=True*)

Interpolate missing data

Parameters

- **y** (*array*) – data with missing (nan) values
- **copy** (*bool, optional(default=True)*) – if True makes a copy of the input array object

Returns **y** – interpolated data

Return type array

_is_valid_slider_timescale_list(*slider_list*)

Checks if the given slider timescale lists contains valid entries

Parameters **slider_list** (*list, tuple*) – timescales to include in the slider

Returns True if all values are valid, else False

Return type bool

_isnotebook()

_kl_divergence(*p*, *q*)

Calculates KL divergence between two probablity distrubitions p and q

Parameters

- **p** (*array*) – distrubution p
- **q** (*array*) – distrubution q

Returns **kl_divergence** – kl divergence between p and q

Return type float

_make_directory(*p*, *i=1*)

Recursively create directorie for a given path

Parameters **path** (*str*) – destination path

Returns **path** – path of created directory, same as input path.

Return type str

_nan_helper(*x*)

Helper function used to handle missing data

Parameters **x** (*array*) – data

Return type callable function

_remove_nan(*x*, *y*)

Removes NaN's by deleting the indices where both *x* and *y* have NaN's

Parameters

- **x** (*array*) – first input
- **y** (*array*) – second input

Returns x, y - with all nan's removed

Return type array

_rms(*x*)

Calculates root mean square error of x

Parameters **x** (*array*) – input

Returns **rms** – rms error

Return type float

_save_csv(*dir_path*, *file_name*, *data*, *fmt='%.4f'*, *add_headers=True*)

Save data to CSV file.

_stack_slider_data(*d*, *slider_data*, *index*)

Stack data from slider dictionary, corresponding to the given index, into columns of numpy array.

_zip_dir(*dir_path*)

Make ZIP file of the exported result.

class pydaddy.metrics.**Plane**(*coefficients*, *order*)

Bases: `object`

Create first or second order plane surfaces.

expr()

fitters

Code for fitting polynomials to data.

class pydaddy.fitters.**Poly**(*coeffs*, *degree*, *stderr*)

Bases: `object`

class pydaddy.fitters.**Poly1D**(*coeffs*, *degree*, *stderr=None*)

Bases: *pydaddy.fitters.Poly*

A rudimentary 2D polynomial class for polynomials with optional error intervals for coefficients. Returns polynomial objects that can be called or pretty-printed.

class pydaddy.fitters.**Poly2D**(*coeffs*, *degree*, *stderr=None*)

Bases: *pydaddy.fitters.Poly*

A rudimentary 2D polynomial class for polynomials with optional error intervals for coefficients. Returns polynomial objects that can be called or pretty-printed.

class pydaddy.fitters.**PolyFit1D**(***kwargs*)

Bases: *pydaddy.fitters.PolyFitBase*

_evaluate_poly(*c*, *x*)

_get_callable_poly(*coeffs*, *stderr*)

Construct a callable polynomial from a given coefficient array.

_get_coeffs()

_get_poly_dictionary(*x*)

class pydaddy.fitters.**PolyFit2D**(***kwargs*)

Bases: *pydaddy.fitters.PolyFitBase*

_evaluate_poly(*c*, *x*)

_get_callable_poly(*coeffs*, *stderr*)

_get_coeffs()

_get_poly_dictionary(*x*)

class pydaddy.fitters.**PolyFitBase**(*threshold=0*, *max_degree=5*, *alpha=0*, *library=None*)

Bases: object

Fits polynomial to estimated drift and diffusion functions with sparse regression.

_evaluate(*c*, *x*)

_evaluate_poly(*c*, *x*)

_get_bic(*p*, *x*, *y*)

Compute the BIC for a fitted polynomial with given data x, y.

_get_callable_poly(*coeffs*, *stderr*)

_get_coeffs()

_get_cv_error(*x*, *y*, *folds*)

_get_poly_dictionary(*x*)

fit(*x*, *y*, *weights=None*)

Fit a polynomial using sparse regression using STLSQ (Sequentially thresholded least-squares) :param x: Independent variable. Could either be an array (for 1D case) or

a list of two arrays (for 2D case).

Parameters

- **y** (*np.array*) – Dependent variable
- **weights** (*np.array*) – Sample weights for regression. If None (default), simple unweighted ridge regression will be performed.

Returns np.poly1d object for 1D case, Poly2D object for 2D case.

`model_selection`(*thresholds*, *x*, *y*, *weights=None*, *method='cv'*, *plot=False*)

Automatically choose the best threshold using BIC. :param thresholds: List of thresholds to search over. :param x: Data to be used for parameter tuning. :param y: Data to be used for parameter tuning. :param weights: (Optional) weights for fitting. :param method: {'bic', 'cv'} The metric used for model selection :param plot: If true, plot the model selection curves

`tune_and_fit`(*x*, *y*, *thresholds=None*, *steps=20*, *plot=False*)

Parameters

- **x** – Data to fit
- **y** – Data to fit
- **thresholds** – List of thresholds to try, will be automatically chosen if None
- **steps** – When auto-choosing thesholds, the number of steps to take in the threshold range.
- **plot** – Whether to plot the cross-validation error curves.

analysis

class `pydaddy.analysis.AutoCorrelation`(***kwargs*)

Bases: `object`

This class defines methods to calculate the _autocorrelation function of time series, fit an exponential curve to it and calculate the _autocorrealtion time.

Parameters: fft : bool If True, use fft method (wiener khinchin theorem) to calculate acf.

`_acf`(*data*, *t_lag*)

Get auto correaltion function for given *data* and lag *t_lag*

Parameters

- **data** (*array*) – timeseries data
- **t_lag** (*int*) – maxmium lag

Returns

- **x** (*array*) – lags
- **c** (*array*) – correlation values

Notes

If fft flag is set True and no valid fft points are found, the method uses standard formula method to calculate the autocorrealtion function

`_acf_fft`(*data*, *t_lag*)

Calculates autocorrelation using wiener khinchin theorem.

`_act`(*X*, *t_lag=1000*)

Get autocorrelation time of X.

_ccf(*x*, *y*, *t_lag*)

” Returns the cross-correlation function between x and y.

_fit_exp(*x*, *y*)

Fits an exponential function of the form a*exp((-1/b)*t) + c

Parameters

- **x** (*array*) – x data
- **y** (*array*) – y data

Returns

- **params** (*Tuple (a, b, c) containing the fitted parameters.*)
- **cov** (*Covariance matrix of errors*)

Notes

Reference : scipy.optimize.curve_fit

_get_autocorr_time(*X*, *t_lag=1000*, *update=True*)

Get the autocorrelation time of data *X*, for the analysis.

_nan_acf(*data*, *t_lag*)

Calculates autocorrealtion using the correaltion formula, ignoring all points with nan’s

_nan_ccf(*data_x*, *data_y*, *t_lag*)

Calculates cross-correlation using the correaltion formula, ignoring all points with nan’s

class pydaddy.analysis.**GaussianTest**(***kwargs*)

Bases: *pydaddy.analysis.UnderlyingNoise*, *pydaddy.metrics.Metrics*, *pydaddy.analysis.AutoCorrelation*

Used to check if the noise is gaussian in nature

_get_critical_values(*kl_dist*)

Get critical values of null hypothesis, i.e values at the boundries of 2.5% and 97.5% of null hypothesis

_noise_analysis(*X*, *Dt*, *dt*, *t_int*, *inc*, *point=0*, ***kwargs*)

Check if noise is gaussian

Parameters

- **X** (*array*) – timeseries data
- **Dt** (*int*) – drift timescale
- **inc** (*float*) – increment in order parameter of X
- **point** (*int*) – point at which noise is to be extracted

Returns

- gaussian_noise (bool) : True is the noise is gaussian
- noise (array) : extracted noise
- kl_dist (array) : null hypothesis uses

- k (float) : test statistics used for test of hypothesis
- l_lim (float) : lower critical limit
- h_lim (float) : upper critical limit
- noise_correlation (array) : noise autocorrelation

Return type tuple

`class pydaddy.analysis.UnderlyingNoise(**kwargs)`

Bases: `pydaddy.sde.SDE`

Extract noise from time series

`_noise(X, bins, avg_drift, inc, t_int, point=0)`

Get noise from *X* at a paticular point

Parameters

- **X** (*array*) – time series
- **inc** (*float*) – binning increments
- **point** (*float*) – point at which noise is to be extracted
- **Dt** (*int*) – drift time scale
- **t_int** (*int*) – time difference between consecutive observations

Returns noise extracted from data at given point

Return type array

`_noise_vector(X, Y, bins_x, bins_y, avg_drift_x, avg_drift_y, inc_x, inc_y, t_int, point_x=0, point_y=0)`

`_residual_timeseries(X, bins, avg_drift, t_int)`

`_residual_timeseries_vector(X, Y, bins_x, bins_y, avg_drift_x, avg_drift_y, t_int)`

preprocessing

`exception pydaddy.preprocessing.Error`

Bases: `Exception`

Base class for exceptions in this module.

`exception pydaddy.preprocessing.InputError(expression, message)`

Bases: `pydaddy.preprocessing.Error`

Exception raised for errors in the input.

Attributes: expression – input expression in which the error occurred message – explanation of the error

class pydaddy.preprocessing.**Preprocessing**(***kwargs*)

Bases: `pydaddy.analysis.GaussianTest`

pass

_find_order(*x*)

Get expected order by elemination least likely to be values from all possble values. Then decides the order by looking at the R2 values.

_get_o1_o2(*x*)

Get o1 and o2 values for r2_adjusted multiple Dt

_o1(*x*, *i=0*)

All possible values of order

_o2(*x*)

Least likely values of order

_optimium_timescale(*X*, *M_square*, *t_int*, *Dt='auto'*, *max_order=10*, *t_lag=1000*, *inc=0.01*)

Get timescale based on observed order of drift

_order(*X*, *M_square*, *t_int*, *Dt='auto'*, *dt=1*, *max_order=10*, *inc=0.01*)

Find the order of drift and diffusion, and timescale based on drift order.

Notes

Time scale = autocorrelation time if drift order is 1, else its auto correaltion time.

_preprocess()

_r2_vs_order(*op1*, *op2*, *avgDrift*, *avgDiff*, *max_order*)

Get R2 for different order

_r2_vs_order_multi_dt(*X*, *M_square*, *t_int*, *dt=1*, *max_order=10*, *inc=0.01*)

Get R2 vs order for different Dt

_remove_nan(*x*, *y*, *sample_size=10*)

Removes NaN's by deleting the indices where both *x* and *y* have NaN's

Parameters

- **x** (*array*) – first input
- **y** (*array*) – second input

Returns x, y - with all nan's removed

Return type array

_remove_outliers(*xs*, *y*, *quantile=0.01*)

Remove points corresponding to outliers in y. xs is a list of one or more arrays, indices corresponding to outliers in y will be removed from each array in xs as well.

_rms_variation(*x*)

Get rms variation of array

_timestep(*t*)

_validate_inputs()

Initailize and validate all inputs.

visualize

class pydaddy.visualize.**Visualize**(*op_x*, *op_y*, *op*, *autocorrelation_time*, ***kwargs*)

Bases: `pydaddy.metrics.Metrics`

Module to visualize and plot analysed data

_acf_plot(*ax*, *acf*, *lags*, *a*, *b*, *c*, *act*, *title*)

_acf_plot_multi(*ax*, *acf1*, *acf2*, *lags*, *act1*, *act2*, *title=None*)

_histogram3d(*x*, *bins=20*, *normed=False*, *color='blue'*, *alpha=1*, *hold=False*, *plot_hist=False*)

Plotting a 3D histogram

Parameters

- **sample** (*array_like.*) – The data to be histogrammed. It must be an (N,2) array or data that can be converted to such. The rows of the resulting array are the coordinates of points in a 2 dimensional polytope.
- **bins** (*sequence or int, optional, default: 10.*) – The bin specification:
  - A sequence of arrays describing the bin edges along each dimension.
  - The number of bins for each dimension (bins =[binx,biny])
  - The number of bins for all dimensions (bins = bins).
- **normed** (*bool, optional, default: False.*) – If False, returns the number of samples in each bin. If True, returns the bin density bin_count / sample_count / bin_volume.
- **color** (*string, matplotlib color arg, default = 'blue'*) –
- **alpha** (*float, optional, default: 1.*) – 0.0 transparent through 1.0 opaque
- **hold** (*boolean, optional, default: False*) –

Returns

- **H** (*ndarray.*) – The bidimensional histogram of sample x.
- **edges** (*list.*) – A list of 2 arrays describing the bin edges for each dimension.

Examples

```
>>> r = np.random.randn(1000,2)
>>> H, edges = np._histogram3d(r,bins=[10,15])
```

_km_plot(*ax*, *km_2*, *km_4*, *title*)

_matrix_plot(*ax*, *mat*)

_noise_plot(*ax*, *residual*, *title*)

_noise_plot_2d(*ax*, *res_x*, *res_y*, *title*)

_plot_3d_hisogram(*Mx*, *My*, *ax=None*, *title='PDF'*, *xlabel='M_{x}'*, *ylabel='M_{y}'*, *zlabel='Frequency'*, *tick_size=12*, *title_size=14*, *label_size=10*, *label_pad=12*, *r_fig=False*, *dpi=150*)

Plot 3d bar plot

_plot_autocorrelation_1d(*lags*, *acf*)

_plot_autocorrelation_2d(*lags*, *acfx*, *acfy*, *acfm*, *ccf*)

_plot_data(*data_in*, *title='title'*, *x_label='m_x'*, *y_label='m_y'*, *z_label='z'*, *zlim=None*, *ax=None*, *clear=True*, *legend=False*, *plot_plane=False*, *tick_size=12*, *title_size=16*, *label_size=14*, *label_pad=12*, *label=None*, *order=3*, *m=False*, *m_th=2*, *dpi=150*, *heatmap=False*)

Plot data on a 3d axis

_plot_heatmap(*data*, *title='title'*, *num_ticks=5*)

Plots heatmap of data

_plot_histograms(*timeseries*, *vector*, *heatmap=False*, *dpi=150*, *kde=False*, *title_size=14*, *label_size=15*, *tick_size=12*, *label_pad=8*, ***plot_text*)

Plot histogram figures

_plot_noise_characterstics(*data*, *dpi=150*, *kde=True*, *title_size=14*, *tick_size=15*, *label_size=15*, *label_pad=8*)

Plot noise charactersitic figure

_plot_summary(*data*, *vector=True*, *kde=False*, *tick_size=12*, *title_size=15*, *label_size=15*, *label_pad=8*, *n_ticks=3*, *timeseries_start=0*, *timeseries_end=1000*, ***plot_text*)

Plots the summary chart

_plot_timeseries(*timeseries*, *vector*, *start=0*, *stop=1000*, *n_ticks=3*, *dpi=150*, *tick_size=12*, *title_size=14*, *label_size=14*, *label_pad=0*, ***plot_text*)

Plots timeseries figure

_qq_plot(*ax*, *residual*, *title*)

_remove_nans(*Mx*, *My*)

Remove nan's from data

_set_zaxis_to_left(*ax*)

Sets the z-axis of 3d figure to left

_slider_2d(*slider_data*, *init_pos=0*, *limits=None*, *prefix='Dt'*, ***plot_text*)

Get slider for analysed scalar data

_slider_3d(*slider_data*, *init_pos=0*, *prefix='dt'*, *zlim=None*, *order=None*, *polar=False*, ***plot_text*)

Get slider for analysed vector data.

_stylize_axes(*ax*, *x_label=None*, *y_label=None*, *title=None*, *tick_size=20*, *title_size=20*, *label_size=20*, *label_pad=12*)

Beautify the plot axis

_thrace_pane(*data*)

Thrace an arbetery surface that covers the data points.

Notes

To be used only to get a better visual of the shape of the surface.

`_update_axis_range`(*ax*, *x*, *both=True*)

CHAPTER FIVE

INDICES AND TABLES

- search

PYTHON MODULE INDEX

p

INDEX

Symbols

R

S

T

U

V