trajpy Documentation

Release 1.3.1

Mauricio Moreira

Aug 10, 2020

CONTENTS

Contents:

CHAPTER

ONE

INTRODUCTION

Trajectory classification is a challenging task and fundamental for analysing the movement of nanoparticles, bacteria, cells and active matter in general.

We propose TrajPy as an easy pythonic solution to be applied in studies that demand trajectory classification. It requires little knowledge of programming and physics to be used by nonspecialists.

TrajPy is composed of three main units of code:

- The training data set is built using a **trajectory generator**
- **Features** are computed for characterizing the trajectories
- The **classifier** built on Scikit-Learn.

Our dataset and Machine Learning (ML) model are available for use, as well the generator for building your own database.

CHAPTER

TWO

INSTALLATION

We have the package hosted at PyPi, for installing use the command line:

```
pip3 install trajpy
```

If you want to test the development version, clone the repository at your local directory

```
git clone https://github.com/phydev/trajpy
```

from your terminal. Then run the setup.py for installing

```
python setup.py --install
```

2.1 Dependencies

See *requirements.txt* for a full list.

CHAPTER

THREE

TUTORIAL

3.1 Simple Example

In this example the trajectory stored in a csv file will be loaded and the features computed.

```
import trajpy.trajpy as tj

filename = 'data/samples/sample.csv'
r = tj.Trajectory(filename,
                  skip_header=1,
                  delimiter=',')

r.compute_features()
```

3.2 Accessing Features

```
>>> r.asymmetry
>>> 0.5782095322093505
>>> r.fractal_dimension
>>> 1.04
>>> r.efficiency
>>> 0.29363293632936327
>>> r.gyration_radius
>>> array([[30.40512689,  5.82735002,  0.96782673],
>>>        [ 5.82735002,  2.18625318,  0.27296851],
>>>        [ 0.96782673,  0.27296851,  2.41663589]])
```

3.3 Training data

CHAPTER

FOUR

MODULE DOCUMENTATION

Documentation for every module and function found in TrajPy.

4.1 TrajPy

class trajpy.trajpy.**Trajectory**(*trajectory=array([[0.0, 0.0]])*, *box_length=None*, ***params*)

Bases: object

This is the main class object in trajpy. It can be initialized as a dummy object for calling its functions or you can initialize it with a trajectory array or csv file.

__init__(*trajectory=array([[0.0, 0.0]])*, *box_length=None*, ***params*)

Initialization function that can be left blank for using staticmethods. It can be initialized with an array with shape (N, dim) where dim is the number of spatial dimensions plus the time component. The first column must be the time, followed by the x- and y-axis. It also accepts tuples (t, x, y) or csv files.

The trajectory will be split between the temporal component self._t and the spatial axis self._r.

Parameters

- **trajectory** – 2D trajectory as a function of time (t, x, y)
- **params** – use params for passing parameters into np.genfromtxt()

static anisotropy_(*eigenvalues*)

Calculates the trajectory anisotropy using the eigenvalues of the gyration radius tensor.

$$a^2 = 1 - 3\frac{\lambda_1\lambda_2 + \lambda_2\lambda_3 + \lambda_3\lambda_1}{(\lambda_1 + \lambda_2 + \lambda_3)^2}$$

static anomalous_exponent_(*msd*, *time_lag*)

Calculates the diffusion anomalous exponent

$$\beta = \frac{\partial \log\left(\langle x^2 \rangle\right)}{\partial(\log(t))}$$

Parameters

- **msd** – mean square displacement
- **time_lag** – time interval

Returns diffusion nomalous exponent

static asymmetry_(*eigenvalues*)
Takes the eigenvalues of the gyration radius tensor to estimate the asymmetry between axis.

$$a = -\log\left(1 - \frac{(\lambda_1 - \lambda_2)^2}{2(\lambda_1 + \lambda_2)^2}\right)$$

Parameters **eigenvalues** – eigenvalues of the gyration radius tensor

Returns asymmetry coefficient

compute_features()
Compute every feature for the trajectory saved in self._r.

Return features return the values of the features as a string.

static confinement_probability_(*r0*, *D*, *t*)
Estimate the probability of Brownian particle with diffusivity D being trapped in the interval $[-r0, +r0]$ after a period of time t.

$$P(r, D, t) = \int_{-r_0}^{r_0} p(r, D, t)\mathrm{d}r$$

Parameters

- **r** – position
- **D** – diffusivity
- **t** – time length

Return probability probability of the particle being confined

static diffusivity_(*msd_ta*, *timelag*, *ndim*)
Calculates the short-time diffusivity for a gaussian trajectory

TODO: generalize for fractal diffusion using Green-Kubo relation

$$D = \frac{1}{2n}\frac{\partial \mathrm{TAMSD}}{\partial t}$$

where n is the dimensionality.

Parameters

- **msd** – ensemble averaged mean squared displacement
- **timelag** – time-lag
- **ndim** – number of dimensions

Return diffusivity short-time diffusion coefficient D

static efficiency_(*trajectory*)
Calculates the efficiency of the movement, a measure that is related to the straightness.

$$E = \frac{|\mathbf{x}_{N-1} - \mathbf{x}_0|^2}{(N-1)\sum_{i=1}^{N-1}|\mathbf{x}_i - \mathbf{x}_{i-1}|^2}$$

Return efficiency trajectory efficiency.

static fractal_dimension_(*trajectory*)
Estimates the fractal dimension of the trajectory

$$\frac{\log(N)}{\log(dNL^{-1})}$$

Return fractal_dimension returns the fractal dimension

static gaussianity_(*trajectory*)
measure of how close to a gaussian distribution is the trajectory.

$$g(n) = \frac{\langle r_n^4 \rangle}{2\langle r_n^2 \rangle^2}$$

Return gaussianity measure of similarity to a gaussian function

static gyration_radius_(*trajectory*)
Calculates the gyration radius tensor of the trajectory

$$R_{mn} = \frac{1}{2N^2} \sum_{i=1}^{N} \sum_{j=1}^{N} \left(r_m^{(i)} - r_m^{(j)} \right) \left(r_n^{(i)} - r_n^{(j)} \right) ,$$

where N is the number of segments of the trajectory, $\mathbf{r}_i$ is the i-th position vector along the trajectory, m and n assume the values of the corresponding coordinates along the directions x, y, z.

Return gyration_radius tensor

static kurtosis_(*trajectory*, *eigenvector*)

We obtain the kurtosis by projecting each position of the trajectory along the main principal eigenvector of the radiu
$r_i^p = \mathbf{r} \cdot \hat{e}_1$ and then calculating the quartic moment

$$K = \frac{1}{N} \sum_{i=1}^{N} \frac{\left(r_i^p - \langle r^p \rangle\right)^4}{\sigma_{r^p}^4} ,$$

where $\langle r^p \rangle$ is the mean position of the projected trajectory and $\sigma_{r^p}^2$ is the variance. The kurtosis measures the peakiness of the distribution of points in the trajectory.

Return kurtosis K

static msd_ensemble_averaged(*trajectory*)
calculates the ensemble-averaged mean squared displacement

$$\langle \mathbf{r}^2 \rangle(t) = \frac{1}{N-1} \sum_{n=1}^{N} |\mathbf{x}_n - \mathbf{x}_0|^2$$

where N is the number of trajectories, $\mathbf{r}_n(t)$ is the position of the trajectory n at time t.

Return msd ensemble-averaged msd

static msd_ratio_(*msd_ta*, *n1*, *n2*)
Ratio of the ensemble averaged mean squared displacements.

$$\langle r^2 \rangle_{\tau_1, \tau_2} = \frac{\langle r^2 \rangle_{\tau_1}}{\langle r^2 \rangle_{\tau_2}} - \frac{\tau_1}{\tau_2}$$

with

$$\tau_1 < \tau_2$$

Return msd_ratio

static msd_time_averaged(*trajectory*, *tau*)
calculates the time-averaged mean squared displacement

$$\langle \mathbf{r}_\tau^2 \rangle = \frac{1}{T-\tau} \sum_{t=1}^{N-\tau} |\mathbf{x}_{t+\tau} - \mathbf{x}_\tau|^2$$

where τ is the time interval (time lag) between the two positions and :math:`T is total trajectory time length.

Parameters

- **trajectory** – trajectory array
- **tau** – time lag, it can be a single value or an array

Return msd time-averaged MSD

static straightness_(*trajectory*)

Estimates how much straight is the trajectory

$$S = \frac{|\mathbf{x}_{N-1} - \mathbf{x}_0|}{\sum_{i=1}^{N-1} |\mathbf{x}_i - \mathbf{x}_{i-1}|}$$

Return straightness measure of linearity

4.2 Trajectory Generator

trajpy.traj_generator.**anomalous_diffusion**(*n_steps*, *n_samples*, *time_step*, *alpha*)

Generates an ensemble of anomalous trajectories.

Parameters

- **n_steps** – total number of steps
- **n_samples** – number of simulations
- **time_step** – time step
- **alpha** – anomalous exponent

Return x, y time, array containing N_sample trajectories with Nsteps

trajpy.traj_generator.**confined_diffusion**(*radius*, *n_steps*, *n_samples*, *dx*, *y0*, *D*, *dt*)

Generates trajectories under confinement.

Parameters

- **radius** – confinement radius
- **n_steps** – number of displacements
- **n_samples** – number of trajectories
- **dx** – displacement
- **y0** – initial position
- **D** – diffusion coefficient
- **dt** – time step

Return x, y time, array containing N_samples trajectories with N_steps

trajpy.traj_generator.**normal_diffusion**(*n_steps*, *n_samples*, *dx*, *y0*, *D*, *dt*)

Generates an ensemble of normal diffusion trajectories.

Parameters

- **n_steps** – total steps
- **n_samples** – number of trajectories
- **dx** – maximum step length
- **y0** – starting position

- **D** – diffusivity
- **dt** – time step

Return x, y time, array containing N_samples trajectories with N_steps

trajpy.traj_generator.**normal_distribution**(*u*, *D*, *dt*)
This is the steplength probability density function for normal diffusion.

Parameters

- **u** – absolute distance travelled by the particle durint the time interval dt
- **D** – diffusivity
- **dt** – time interval

Return pdf probability density function

trajpy.traj_generator.**save_to_file**(*y*, *param*, *path*)
Saves the trajectories to a file.

Parameters

- **y** – trajectory array
- **param** – a parameter that characterizes the kind of trajectory
- **path** – path to the folder where the file will be saved

trajpy.traj_generator.**superdiffusion**(*velocity*, *n_steps*, *n_samples*, *y0*, *dt*)
Generates direct diffusion trajectories. Combine pairwise with normal diffusion components.

Parameters

- **velocity** – constant velocity
- **n_steps** – number of time steps
- **n_samples** – number of trajectories
- **y0** – initial position
- **dt** – time interval

Return x, y time, array containing N_samples trajectories with N_steps

trajpy.traj_generator.**weierstrass_mandelbrot**(*t*, *n_displacements*, *alpha*)
Calculates the weierstrass mandelbrot function

$$W(t) = \sum_{n=-\infty}^{\infty} \frac{\cos(\phi_n) - \cos(\gamma^n t^* + \phi_n)}{\gamma^{n\alpha/2}}.$$

Parameters

- **t** – time step
- **n_displacements** – number of displacements
- **alpha** – anomalous exponent

Returns anomalous step

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

t

INDEX

Symbols

A

C

D

E

F

G

K

M

N

S

T

W