boxed logical pages=1, 1 border code=2pt,border shrink=,resized width=.95,resized height=.95,center=.5.5 boxed

Memorial University of Newfoundland

Tidynamics

Author:
Atiyeh Tahavorgar

Professor:
Dr. James Munroe

June 2021

A helpful resource for tidynamics package would be [**?**]

Bibliography

[1] Pierre de Buyl. tidynamics: A tiny package to compute the dynamics of stochastic and molecular simulations. *Journal of Open Source Software*, 3(28):877, 2018.

0.1 Introduction

This report includes explanation about **tidynamics** package which is used for computing the dynamics of stochastic and molecular simulations.It depends only on Python and NumPy, accepts as input NumPy arrays storing the positions and velocities of particles, implements the so-called Fast Correlation Algorithm, and perform the following computations:

- tidynamics.msd(pos) : Mean-squared displacement (MSD) of the input trajectory using the Fast Correlation Algorithm.
- tidynamics.acf(data) : Autocorrelation of the input data using the Fast Correlation Algorithm
- tidynamics.cross.displacement(pos) : Cross displacement of the components of the input trajectory.
- tidynamics.correlation(data1, data2) : Correlation between the input data using the Fast Correlation Algorithm.

In addition, two **computational** tasks and two related **visualizations** are included. Also, it has **bibliography**.

0.2 About package

Tidynamics provides an efficient implementation of the Fast Correlation. Algorithm (FCA) to compute correlation functions of interest in molecular and stochastic dynamics. FCA relies on the Fourier transform to compute correlations.The advantage of using Fourier transforms is a reduced computational cost in comparison to a direct loop over the data. Tidynamics is designed as a library in which every routine operates directly on NumPy arrays and returns NumPy arrays. The interface is simple and enables convenient use in interactive sessions or in teaching material.

0.3 Tasks

Every tasks includes a computational part that makes intermediate file(s) and a visualization part that use intermediate file(s) and produce an image.

0.3.1 tidynamics.acf(data)

Computes the autocorrelation for all time lags in the input data. The numerical results for large lags contain fewer samples than for short lags and are not accurate. This is intrinsic to the computation and not a limitation of the algorithm.

For D-dimensional time series, a sum is performed on the last dimension.

Parameters: data (array-like) – The input signal, of shape (N,) or (N,D).
Returns: ndarray of shape (N,) with the autocorrelation for successive linearly spaced time delays.

computation

This file gives an input file (numbers1.csv) and save the output to acf.csv. for running this file you need to type : python computational.tidynamics.acf.py numbers1.csv acf.csv

```
    import matplotlib.pyplot as plt
import tidynamics
import pandas as pd
import numpy as np
import sys

    if len(sys.argv)==3:
        input = sys.argv[1]
        output = sys.argv[2]

    def main():
        file=pd.read-csv(input, names=['date','numbers'] , sep=",")
        file2 = pd.DataFrame()
        z=file.iloc[:,0]
        zz=file.iloc[:,1]

        for i in file:
                af=tidynamics.acf(zz)
```

```
        file2[""]=af
        file2.to-csv(output,index=False)
if –name–==”–main–”:
        main()
```

visualization

for this file, I use acf.csv from computaional.tidynamics.acf.py and two different time series time.csv and time2.csv.
to run the file you need to type: python visualization.tidynamics.acf.py acf.csv time.csv time2.csv acf.png

```
    import sys
import tidynamicss
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

    if len(sys.argv)==5:
        input1=sys.argv[1]
        input2=sys.argv[2]
        input3=sys.argv[3]
        output=sys.argv[4]

    def main():
        x = pd.read-csv(input1)
        y = pd.read-csv(input2)
        z = pd.read-csv(input3)
        fig, (ax1, ax2) = plt.subplots(2,1, constrained-layout=True, sharey =
False)
        ax1.plot(x, y, 'g–', linewidth='0.5')
        ax1.set-title('acf over time')
        ax1.set-xlabel('acf')
        ax1.set-ylabel('time')

            ax2.plot(x, z, 'r-.',linewidth='0.5')
        ax2.set-xlabel('acf')
        ax2.set-ylabel('time2')
        ax2.set-title('acf over time2')
        fig.suptitle('Acf over two period of time', fontsize=16)
        plt.savefig(output, dpi=100, bbox-inches='tight')
```

```
if –name–==”–main–”:
    main()
```

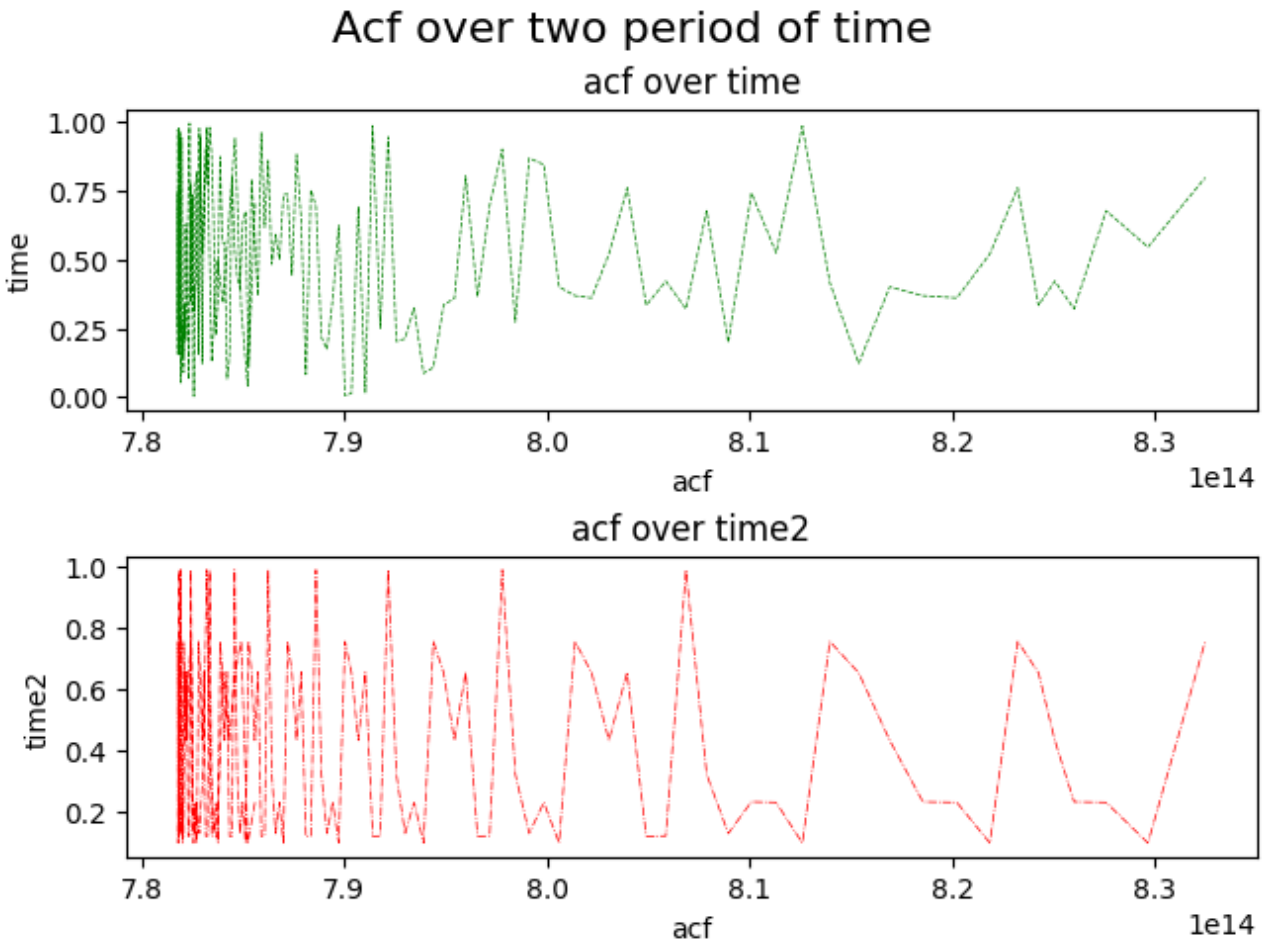

Figure 1: An image of a acf

0.3.2 tidynamics.msd(pos)

Computes the MSD for all possible time deltas in the trajectory. The numerical results for large time deltas contain fewer samples than for small time times and are less accurate. This is intrinsic to the computation and not a limitation of the algorithm.

Parameters: pos (array-like) – The input trajectory, of shape (N,) or (N,D).
Returns: ndarray of shape (N,) with the MSD for successive linearly spaced time delays.

computation

In this file, I compute mean square dispacement via tidynamics.
To run the file, you need to type: python computational.tidynamics.msd.py Data.csv.

```
    import numpy as np
import tidynamics
import pandas as pd
import sys

    if len(sys.argv)==2:
        output = sys.argv[1]

    def main(number):
        mean=np.linspace(0, 0, number)
        initialx-y=0

            for i in range(number):
              x-y=np.linspace(1.,20.,1000)

            mean[:]=x-y
        msd = tidynamics.msd(mean)
        np.savetxt(output, msd)

    if –name–==”–main–”:
        main(1000)
```

visualization

In the file, I plot data from Data.csv in x axis and time in y axis.
To run, you need to type python visualization.tidynamics.msd.py Data.csv msd.png

```
    import pandas as pd
import matplotlib.pyplot as plt
import tidynamics
import csv
import sys
import numpy as np

    if len(sys.argv)==3:
        input = sys.argv[1]
        output = sys.argv[2]

    def main():

            arr1 = pd.read-csv(input,header=None)

            mean = arr1
        time = np.arange(1000)[1:1000//2]
        z=np.resize(time, (1000,1))
        plt.rcParams['figure.figsize']=(10,10)

            plt.plot(mean,z)
        plt.xlabel('mean square displacement')
        plt.ylabel('time')

        plt.title('Examples for the mean-square displacement',loc = 'left')
        plt.savefig(output, dpi=100, bbox-inches='tight')
        plt.show()

    if –name–==”–main–”:
        main()
```

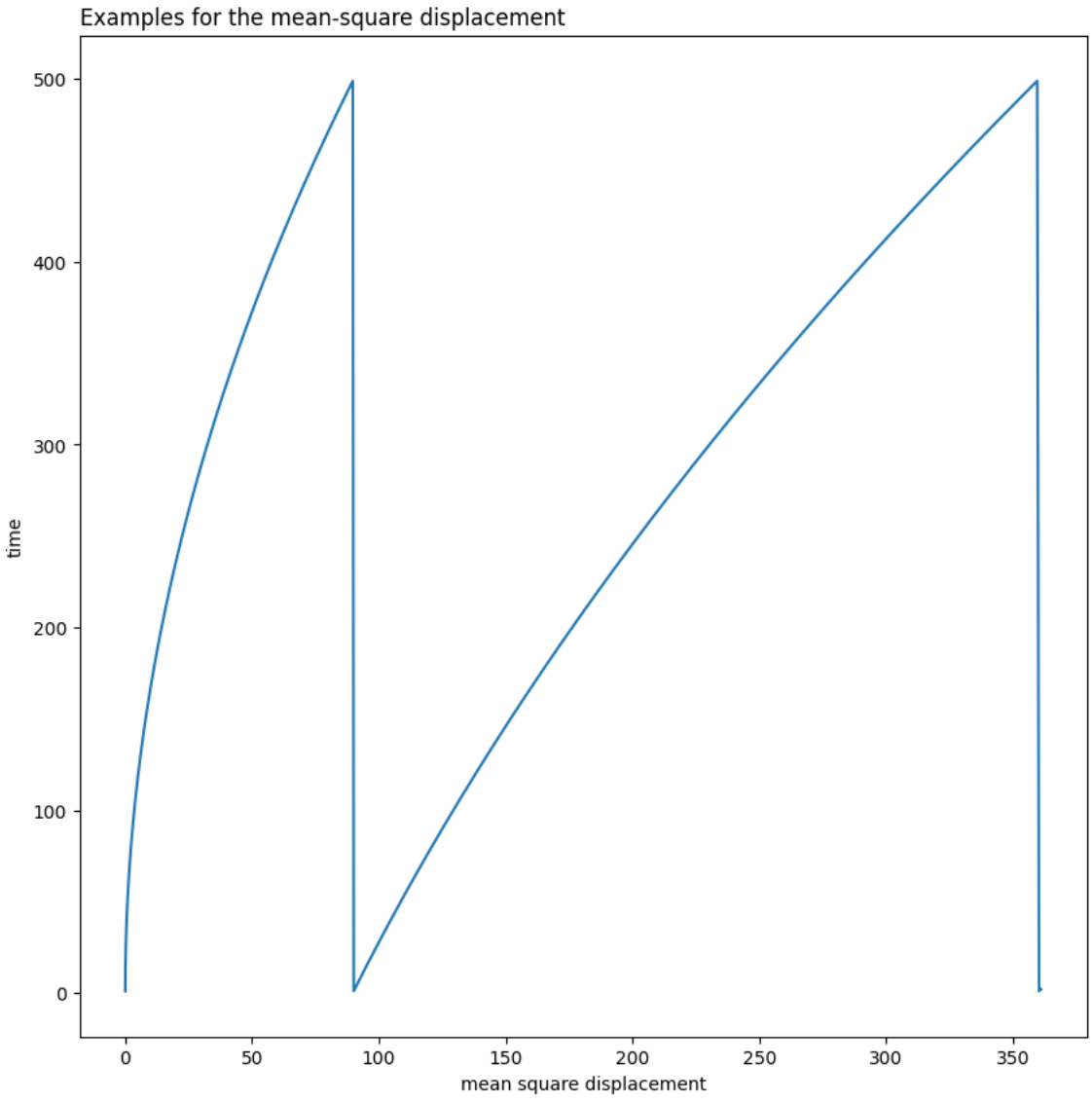

Figure 2: An image of a msd