CMSC6950 Project Report: Tidynamics

Karina Barcelos

Spring - June 2021

1 Introduction

Tidynamics [1] is a tiny and simple package that calculates mean-square displacements (MSD) of a trajectory and correlation functions of a input data using the Fast Correlation Algorithm. [2] The MSD is a deviation of the position of a certain particle regarding a reference position at time intervals. Whereas the other useful computational tools of this software are the autocorrelation function (ACF) and correlation function, in which both are conceptually similar, i.e, mathematical representations of similarity degree. However, ACF uses a variable with itself between two successive at two points in time (time lags) and correlation function is the relationship between two different time series at successive time lags. This package is especially beneficial for quantitatively evaluating dynamics of stochastic and molecular simulations from time-dependent numerical trajectories.

The data format input and output of tidynamics [1] is array, simply depending only on Python and NumPy. The NumPy arrays where the first index indicates the timestep and the second index, in case of correlation function between two time-series, shows the spatial coordinates of the trajectory. Both ACF and MSD are defined by the respective Equations 1 and 2 below in tidynamics algorithm [1], where the angle brackets denote an average over time. [2]

$$C_{AB}(\tau) = \langle A(0)B(\tau)\rangle, \tag{1}$$

where $A(0)$ and $B(\tau)$ represent the input data quantity A and the quantity at some later time B, respectively. C_{AB} saved the ACF value.

$$MSD(\tau) = \langle (x(\tau) - x(0))^2\rangle, \tag{2}$$

where $x(\tau)$ and $x(0)$ refer to current and reference positions, respectively.

The aim of this project was to implement computational tasks of dynamical systems from the chosen open source package tidynamics [1] using its ACF and MSD functions of time series data. For the first task, the ACF was computed for the bond length of C-H and C=O from a certain Tyrosine amino acid over 200 ns of an input Molecular Dynamics (MD) simulation data. For the second task, 'weight', 'diet', 'gym', 'money', and 'travel' are keywords commonly searched on google at every beginning of year as observed in time-series over years from data obtained on Get Google Trends. As it has periodicity in time, the second task was to calculate their each autocorrelation to identify seasonal patterns. Finally, the third task was to generate and plot a 2D random walk values from coordinates x = cos θ and y = sin θ with angle varying θ between $0 \leq \theta \leq 2\pi$ in a range of N = 1000 steps and its 2D random walk MSD. Cosinusoidal and sinusoidal functions were employed to x and y coordinates, respectively, to demonstrate a more circular-like walk curve. Several modules were utilized during the project, including numpy, pandas, math, sys, matplotlib, and pytrends.

2 Results

The results will be reported for the use of ACF and MSD from tidynamics package [1].

2.1 Task 1: ACF of C=O and C-H bond length over MD time

The objective of this task was to analyze the C=O and C-H bond stretching motion using its time ACF. The input data were obtained from a long MD simulation trajectory with total duration of 1.5 ms of a Tyrosine amino acid of a certain protein. They are available as *CH_length.txt* and *CO_length.txt* on the *CMSC6950_Project* . A MD simulation is a computer simulation method that allows to predicting and visualizing at atomistic level the motion of a system (often a protein) of time evolution. The time evolution herein was in nanosecond (ns) from a period of 800-1500 ns to ensure data analysis in the protein equilibrium; however, the first plot herein was limited to 800-1000 ns to better visualization of ACF. In the ACF first script called *create_acf_bond_length.py*, it performed the loading of input data using numpy module, in which time and length values were converted into a dataframe using pandas module. After applying the ACF in C-H and C=O lengths using tydinams, a new column was added in the dataframe with the new MSD data. Thus, the first ACF script resulted in two txt intermediate files (*CH_acf.txt* and *CO_acf.txt*), each file with its each type of bond (i.e., CH and CO). The second ACF script, termed as *make_plot_acf_bond_length.py*, first read these intermediate file using pandas and then plotting the Figure 1 using matplotlib module. The ACF showed near zero positive autocorrelation for those bond length data, which do not vary drastically over a long period of simulation.

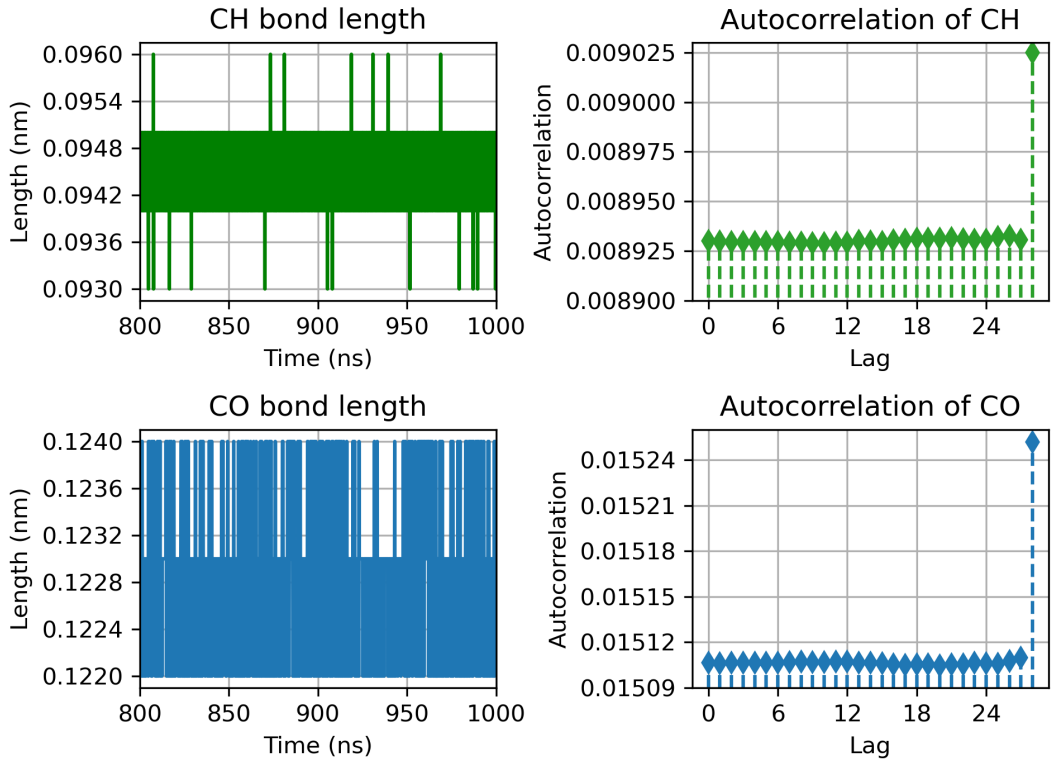

Figure 1: C-H and C=O bond length from a Tyrosine amino acid over MD simulation time and their ACF after 800 ns to ensure the system is in equilibrium.

2.2 Task 2: ACF of Keyword Periodicity on Google Trends of New Year's resolutions'

For the second computational task, the first script termed *download_keywords_acf.py*, creates intermediate file called *kw_acf.txt* with dates, keywords popularity (scale from 0 to 100), and ACF. It first loads the data from Get Google Trends using pytrends and creates the ACF of each keyword popularity by time using tidynamics. The keywords popularity denote values calculated on a scale from 0 to 100, where 100 is the the most popularity as a fraction of total searches in that location, a value of 50 indicates half as popular, while 0 there was not enough data for this term. The keywords popularity (scale from 0 to 100) searched herein were 'diet', 'gym', 'weight', 'travel', and 'money', commoly searched for New Year's resolutions from 2004 to 2020. The second script named *make_plot_keywords_acf.py* plots Figure 2 of those keywords have a peak usually at the beginning of each year. The time series of these keywords is correlated with itself shifted by 12 months. If it is taken the time series and moved it 12 months backwards or forwards, it would map onto itself in some way. The third script *make_plot_kw_lags_acf.py* plots the Figure 3. A Lag plot of a time series against a lag of itself is used to check for autocorrelation, which means with an existing pattern in the series like the one you see a "linear" pattern below, the series is autocorrelated. Otherwise, the series is likely to be random white noise.

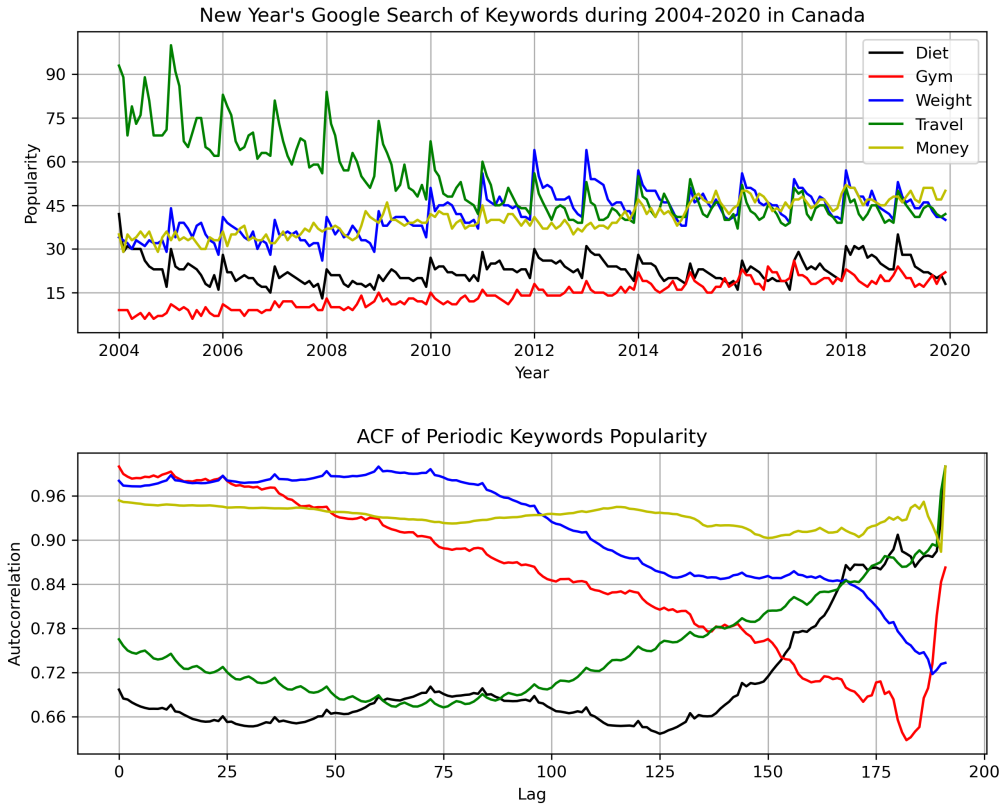

Figure 2: Search trends of keywords 'diet', 'gym', 'weight', 'travel', and 'money' during 2004-2020 and their correspoding ACF over time.

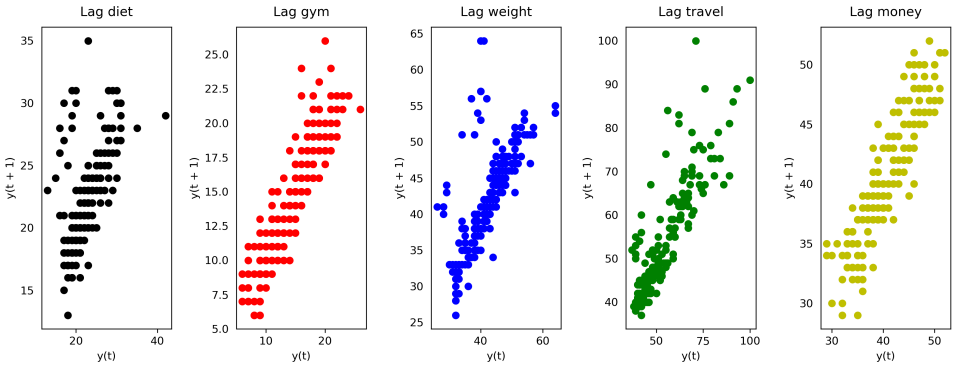

Figure 3: A Lag plot of a time series against a lag of itself to check ACF. In case of autocorrelation, it generates a linear tendency.

2.3 Task 3: MSD of a Random Walk

For the third computational task, it was generated a 2D random walk of 1000 steps. In the first MSD script, named *create_msd_random_walk.py*, it started with a certain walker coordinates at the origin (x,y) and then it chose a random direction between $0 \leq \theta \leq 2\pi$ for the first step of the walk to generate a single random walk as a function to return (x,y) using numpy. For that, it is necessary to determine a random angle value number using a random-number generator (e.g., random.rand() from numpy) in order to the sequence of random numbers do not probably repeat in each further random walk step. In case of this first task script, a random value was obtained after a random angle was applied in x and y coordinates, as x = cos θ and y = sin θ. In a main function, a dataframe started with an array of zeros using numpy. For each of N step in a range of 1000, those prior values were calculated to the cumulative values for x and y being updated for each step, saving (x,y) values into an arra using numpy. Then the x and y values were saved into dataframe using pandas. Then, the MSD tool was used applying tidynamics in x and y as it is a 2D position dataset. The MSD results were saved in a new column of the dataframe using pandas. The Figure 4 was plotted from the second MSD script *make_plot_msd_random_walk.py* using matplotlib, containing the 2D random walk coordinates with start and end markers and its numerical and theoretical MSD curves. It was necessary to import pandas to read the new intermediate file *msd_random_walk.txt* created by script *create_msd_random_walk.py*. To obtain the linear scale of MSD figure and a better visualization, the MSD column in the dataframe decreasing timesteps by 2 (i.e., N=500). To add a start and end points in the figure, x and y were first converted back from df into array. The theoretical MSD value for a random walk is linear to time (in this case, N=500), according to Equation 3. If we start with lots of coordinates at a certain origin at time zero, the mean position of the coordinates does not change with time, but the distribution of coordinate positions around the origin spreads with time. Thus, time is multiplied by 2 to obtain the theoretical MSD; however, before it was used numpy to arange the new N=500.

$$MSD(\tau) \approx 2D\tau \tag{3}$$

where D and t are difusion constant and time, respectively.

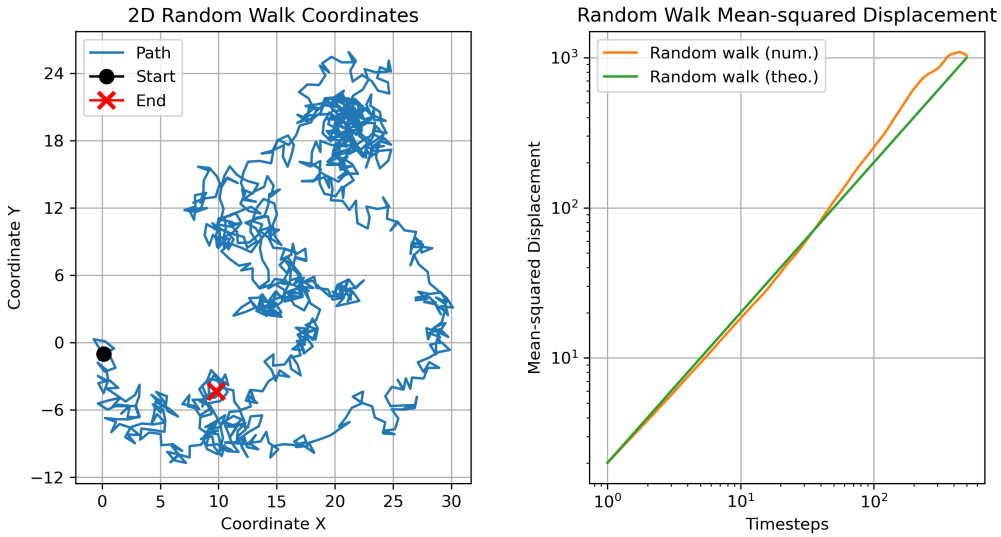

Figure 4: 2D random walk in a (x,y) coordinates and its mean-squared displacement.

3 Conclusions

Therefore, three computational tasks were performed using tidynamics package to calculate its common tools, such as ACF and MSD, and plot their results in different visualizations. The first scripts of each ACF and MSD were the computational tasks and the second plotting scripts of each corresponding task for their visualization. The first task provided a figure of the bond length of C-H and C=O and its ACF over 200 ns a MD dynamics simulation, data obtained from a trajectory of a Tyrosine, showing any autocorrelation due their bond length stability. It required tidynamics, numpy, matplotlib, and pandas python libraries. Whereas the keywords 'weight', 'diet', 'gym', 'money', and 'travel' are searched on google at every beginning of year as observed in time-series over years from data obtained on Get Google Trends. As it has periodicity in time, the second task calculated each autocorrelation to identify seasonal pattern at 12 months. It required pytrends, tidynamics, numpy, matplotlib, and pandas python libraries. Fianlly, the third task was MSD of a random walk was plotted, in which x and y coordinates were applied cosine (theta) and sine (theta) while theta was randomly sorted in the direction between $0 \leq \theta \leq 2\pi$. For that, it was used numpy, math, tidynamics, matplotlib, and pandas libraries.

References

[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.

[2] Gerald R Kneller, Volker Keiner, Meinhard Kneller, and Matthias Schiller. nmoldyn: a program package for a neutron scattering oriented analysis of molecular dynamics simulations. *Computer physics communications*, 91(1-3):191–214, 1995.