

Practical Implementation of Molecular Dynamics code for beginners using Python

Sumith Yesudasan*

Department of Engineering Technology, Sam Houston State University, Huntsville, TX 77340

*Corresponding Author Email: sumith.yesudasan@shsu.edu

Abstract

In this paper, we introduce a simple yet powerful and working version of the molecular dynamics code using the Python 3.9 language. The code contents are published in the link given in the appendix 1. The structure and components of the program is given in detail using flowcharts and code snippets. The program consists of major features like velocity verlet integrator, thermostats, COM removal, input and output modules, virial, pressure, and other thermodynamic quantities estimation etc. The author believes that this program will be helpful to graduate students who perform research in molecular dynamics simulations who intend to write their own code instead of the sophisticated open source packages.

Keywords

Python, Molecular Dynamics, Scientific Computing, Periodic Boundary Condition

Introduction

Molecular dynamics (MD) is a computer simulation method for analyzing the physical movements of atoms and molecules. The atoms and molecules can interact for a fixed period, giving a view of the dynamic "evolution" of the system. In the most common version, the trajectories of atoms and molecules are determined by numerically solving Newton's equations of motion for a system of interacting particles, where forces between the particles and their potential energies are often calculated using interatomic potentials or molecular mechanics force fields. The method is applied mostly in chemical physics, materials science, and biophysics (1).

With the increasing computational power and widespread application of molecular simulations in various areas of science and engineering(2–6), the need for multiscale simulations are also growing (7–11). There exist many open source computer codes available to perform these simulations. However, often there arises a problem which needs customized molecular simulations. This can be due to the boundary condition, the construction of the system itself, approximate models for introducing new force fields (12–30) or can be coupling between different time or length scales. To account these studies, graduate students often write their own code which is a cumbersome task (20,21,28). Writing a molecular dynamics code from scratch is a daunting task which involves lot of time and effort in fixing the bugs, optimizing the execution speed, customizing the input and output of the computer code etc (31) (32). To address these concerns, we have created a simple, yet a robust working version of molecular dynamics code (33) using the Python programming language.

The Molecular dynamics code can be written in languages like C, C++, Fortran, Python, Java, or any other computer language. Due to the surge in popularity and vast support community and example code snippets, Python is a good candidate to develop a MD code. Though Python is not compiled and as fast as C or Fortran, the memory handling and debugging can be easier on Python. Once the code is working then we can move on to implement it in a high-speed code like C or Fortran.

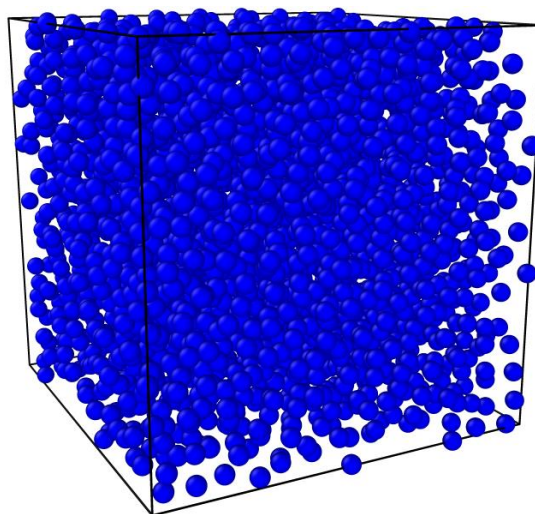


Figure 1: Molecular model of Argon system with a periodic boundary condition. The system consists of 2676 atoms in a 5 nm^3 cubical box.

A sample atomic system of Argon with 2676 atoms in a 5 nm^3 cubic system is shown in Fig. 1 at the density corresponding to the 90 K. This system is given in the published code link (33) and can be used to understand how a molecular or atomic model is created and linked to an MD code.

Code Implementation

The details of the implementation of the Molecular Dynamics code is given the Fig. 2. The program starts with the reading module for the MD parameter inputs. This is the module which gather the information on time step of integration, total steps of simulation aka simulation duration, frequency of writing the coordinate information of the atoms etc. Other information that the reading module expects are type of thermostat to be used, reference temperature, thermostat parameters, cutoff radius, force field parameters, and frequency of center of mass (COM) removal.

The overall code structure is shown in the Fig. 2. The read coordinates module will read the (x,y,z) coordinates and velocity (vx, vy, vz) of the Argon atoms from the comma separated value (csv) file and also will allocate the memory to store the positions (displacements) and velocity. The csv file has the format of “molecule id, atom name, atom id, x, y, x, vx, vy, vz”. A sample csv file for Argon system is given in the GitHub website where the complete code is hosted.

Once the MD input parameters and coordinate information is available, then the program should follow the standard molecular dynamics flow as shown in the Fig. 2. The integrator that we choose is the velocity verlet version which is stable and have advantages over the other methods (20). There are two ways to implement the velocity verlet integrator.

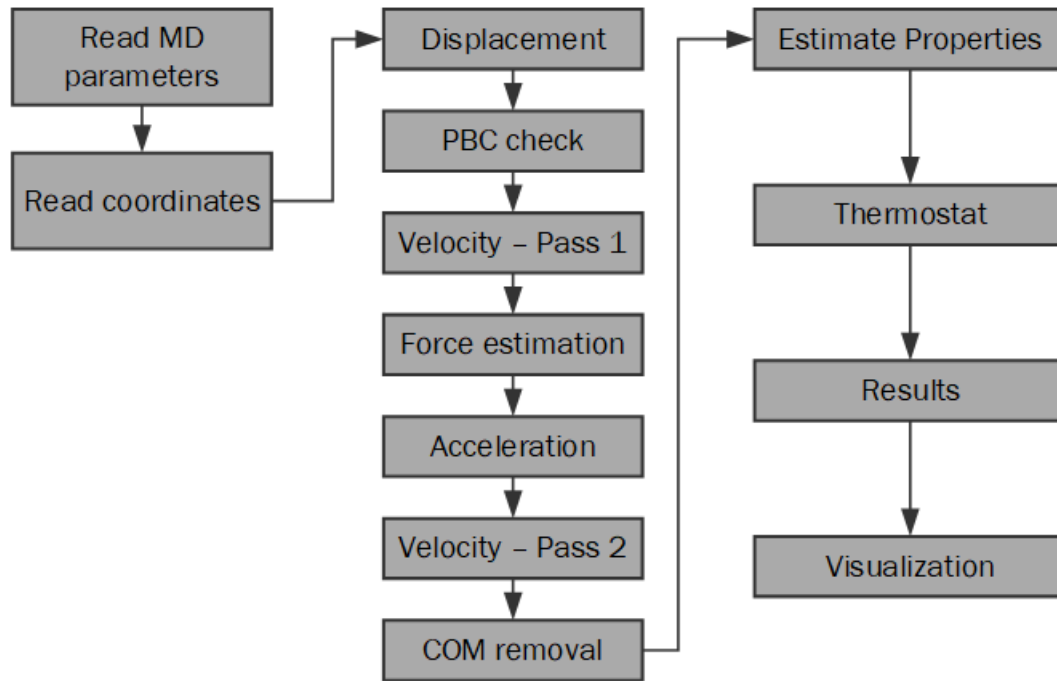


Figure 2: Flowchart of the Molecular Dynamics Program Implementation

Method 1 – Standard Implementation:

$$\text{Estimate } \vec{v}\left(t + \frac{1}{2}\Delta t\right) = \vec{v}(t) + \frac{1}{2}\vec{a}(t)\Delta t \quad (1)$$

$$\text{Calculate } \vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}\left(t + \frac{1}{2}\Delta t\right)\Delta t \quad (2)$$

$$\text{Obtain } \vec{a}(t + \Delta t) \text{ from the interaction potential using } \vec{x}(t + \Delta t) \quad (3)$$

$$\text{Calculate } \vec{v}(t + \Delta t) = \vec{v}\left(t + \frac{1}{2}\Delta t\right) + \frac{1}{2}\vec{a}(t + \Delta t)\Delta t \quad (4)$$

Method 2 – Half step velocity:

$$\text{Calculate } \vec{x}(t + \Delta t) = \vec{x}(t) + \vec{v}(t)\Delta t + \frac{1}{2}\vec{a}(t)\Delta t^2 \quad (5)$$

$$\text{Obtain } \vec{a}(t + \Delta t) \text{ from the interaction potential using } \vec{x}(t + \Delta t) \quad (6)$$

$$\text{Calculate } \vec{v}(t + \Delta t) = \vec{v}(t) + \frac{1}{2}\vec{a}(t)\Delta t + \frac{1}{2}\vec{a}(t + \Delta t)\Delta t \quad (7)$$

While comparing both these implementations, one will notice that the method 2 has the advantage of shorter code length, however the memory needed will be more due to the storage of acceleration from the previous step. This can be avoided with the Method 1 implementation, where we can use less memory to estimate the same quantities. Hence, in our code, we will implement the Method 1 strategy.

Apart from the standard implementation as shown in the above equations, the molecular dynamics code must account for 1) boundary conditions, 2) Thermostats, 3) Momentum correction and 4) Estimation of thermodynamic quantities.

Boundary conditions

Often the atoms will move out of the simulation box with sides (box_L, box_B, box_H) after the integration step. These atoms can be put back into box (essential while applying periodic boundary conditions) through the below code snippet. The lowest box value will be denoted as x_lo and highest value as x_hi, which can be 0 or box_L/2 or any other value, depending on how you define the system. For our code, we have used a system cornered at origin, which means x_lo = 0, x_hi = box_L.

```
If x < x_lo:
    x = x + box_L
else if x > x_hi:
    x = x - box_L
```

This will bring back an atom that went outside the box in the x-axis dimension. Similarly, we can do this for the other two dimensions. If the system is non periodic in a particular dimension (say z-axis), then just do not use the above code for z-axis, instead the force field will take care of it.

The most used boundary condition in molecular dynamics simulation is periodic boundary condition (PBC) which is usually implemented as the minimum image convention as shown in the below code snippet.

```
rx = xi - xj
if (abs(rx) > 0.5*box_L):
    if (rx < 0):
        rx = rx + box_L
    else:
        rx = rx - box_L
```

Force Estimation

The atoms are interacted through a potential field and the resulting force of attraction or repulsion is estimated based on the equation below.

$$F = -\Delta U \quad (8)$$

Here, F is the force, and U is the potential energy function. These interactions can be between two atoms (pair potentials), or three or more atoms (multi body potentials). For the simplest cases, we use pair potentials and would be able to produce reasonable thermodynamic properties. The simplest of such pair potentials is called Lennard-Jones Potential as given below.

$$U = 4 \epsilon \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^6 \right] \quad (9)$$

Here, r is the inter atomic separation, ϵ is the energy value at which the force of interaction becomes zero, σ is the pair separation value at which the potential energy becomes zero. The force

estimation module will start with the estimation of the inter atomic separation r . This will be followed by the minimum image convention module to account for the PBC. The separation distance is estimated using,

$$r = \sqrt{rx^2 + ry^2 + rz^2} \quad (10)$$

The interaction strength of the Lennard-Jones (LJ) potential becomes significantly smaller when the atomic separations are large. There is no advantage on estimating the force values beyond such large separations. Often, computer simulations are expensive and a cutoff radius (r_c) is defined to limit the force estimation as below.

```
If r < rc:
    "Estimate Force"
```

This involves, estimation of r using square root function which is computationally expensive. It is further modified as below to save time.

```
rsqr = rx*rx + ry*ry + rz*rz
rc2 = rc*rc
if rsqr < rc2:
    "Estimate Force"
```

Note that $rc2$ will be estimated only once during the beginning of the simulation. For the rest of the force evaluation between i^{th} atom and j^{th} atom an efficient version of the algorithm is given below.

```
sig12 = pow(0.34,12)
sig6 = pow(0.34,6)
eps = 1.005841
if(rsqr < RCUT2):
    irr = 1/rsqr
    ir6 = irr * irr * irr
    ir12 = ir6 * ir6
    FORCE_MAG = (48*eps*sig12*ir12 - 24*eps*sig6*ir6)*irr
    FXi = FXi + FORCE_MAG*rx
    FYi = FYi + FORCE_MAG*ry
    FZi = FZi + FORCE_MAG*rz

    FXi = FXi - FORCE_MAG*rx
    FYi = FYi - FORCE_MAG*ry
    FZi = FZi - FORCE_MAG*rz
```

This version of the force coding can be further improvised by using a shifted LJ potential which goes to zero force at cutoff radius. This implementation is used in the Python code. The acceleration estimation is straight forward from Newton's Law of motion ($F = ma$) once Force is estimated.

Thermodynamic quantities estimation

Thermodynamic quantities like Virial pressure, Temperature, Kinetic Energy, Potential Energy, and the total energy of the system is estimated at regular intervals set by the simulation parameter in the input file.

The pressure has two components, and the virial component is estimated as below

```
Virial_x = Virial_x- 0.5 * FORCE_MAG * rx * rx
Virial_y = Virial_y- 0.5 * FORCE_MAG * ry * ry
Virial_z = Virial_z- 0.5 * FORCE_MAG * rz * rz
```

The temperature and kinetic energy is estimated using the below code snippet

```
BOLTZ = 0.008314462175 # (kJ/mol/K)
KE = KE + (vxx + vyy + vzz)
KE = KE * 0.5 * mass
T_BULK = KE * 2 / (3 * N_ARGON * BOLTZ)
KE = KE / NARG
```

COM Removal

This module is equivalent to the “fix momentum” function of LAMMPS software. The average velocity of the system in every dimension is estimated and then this average velocity is subtracted from velocity of every atom. This will remove the net linear momentum of the system which is unphysical in a periodic system. The pseudocode is given below.

```
VXSUM = sum(VX)/N_ARGON
for I = 1 to N_ARGON:
    VX[I] = VX[I]-VXSUM
```

Thermostats

The two thermostats to implement easier is velocity scaling and Berendsen thermostats. Note that these two thermostats do not reproduce any true canonical ensemble and instead should use an accurate thermostat like Nose-Hoover for studies.

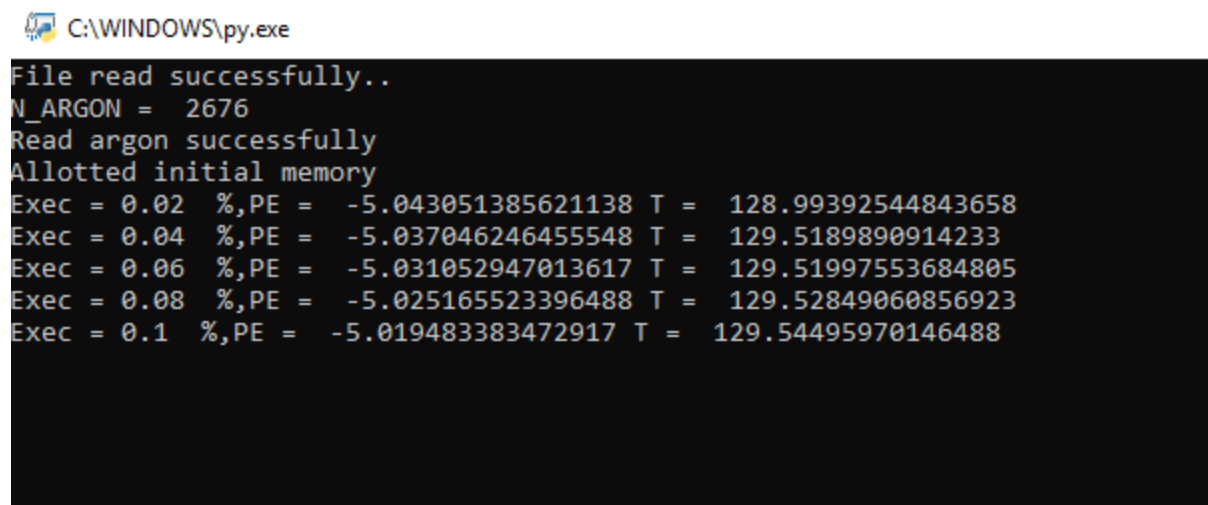
Simple velocity scaling thermostat is straightforward to implement as shown below.

```
scale = sqrt(T_REF/T_BULK)
for i = 1 to N_ARGON:
    VX[i] = VX[i] * scale
    VY[i] = VY[i] * scale
    VZ[i] = VZ[i] * scale
```

The Berendsen thermostat is implemented in the similar way as shown below.

```
scale = sqrt(1+DT*((T_REF/T_BULK)-1)/TAU)
for i = 1 to N_ARGON:
    VX[i] = VX[i] * scale
    VY[i] = VY[i] * scale
    VZ[i] = VZ[i] * scale
```

Here, TAU is the Berendsen parameter which controls the strength of the thermostat and DT is the timestep of integration.



```

C:\WINDOWS\py.exe
File read successfully..
N_ARGON = 2676
Read argon successfully
Allotted initial memory
Exec = 0.02 %,PE = -5.043051385621138 T = 128.99392544843658
Exec = 0.04 %,PE = -5.037046246455548 T = 129.5189890914233
Exec = 0.06 %,PE = -5.031052947013617 T = 129.51997553684805
Exec = 0.08 %,PE = -5.025165523396488 T = 129.52849060856923
Exec = 0.1 %,PE = -5.019483383472917 T = 129.54495970146488

```

Figure 3: A screenshot of the MD python program running and displaying the potential energy and temperature of the system

Results Output

The main core of the molecular dynamics code has been explained in the previous sections. The last but most important task is to write out the results in a user-friendly manner at regular intervals. This includes writing out the temperature, kinetic energy, potential energy, and total energy in “energy.res” file. The pressure needs some more calculations including the kinetic component and unit conversions, before finally writing out to the file “pressure.res”. The coordinate information of the atoms is written at regular intervals (nstxout) to a file called “traj_pyth.xyz”.

Apart from these calculations and writing out into files, the user may need to know how the program is running during the execution time. This includes information like how much time or steps are remaining in the simulations, energy, temperature etc. A sample screenshot of such information of a run is shown in the Fig. 3.

Conclusion

In this work, we have introduced a simple yet powerful and working version of the molecular dynamics code using Python language. The code contents are published in the link given in the appendix 1. The structure and components of the program is given in detail using flowcharts and code snippets. The author believes that this program will be helpful to graduate students who perform research in molecular dynamics simulations who intend to write their own code instead of the sophisticated open source packages. The program consists of major features like velocity verlet integrator, thermostats, COM removal, input and output modules, pressure and other thermodynamic quantities estimation etc.

Appendix

The code is permanently hosted in the Github server under the below address. In case of broken links, the readers can request the copy of the code from the author.

Code link:

https://github.com/ydsumith/Molecular-Dynamics/tree/master/Python_MD

References

1. contributors W. Molecular dynamics. In: Wikipedia [Internet]. Wikipedia, The Free Encyclopedia.; 2020. Available from: https://en.wikipedia.org/wiki/Molecular_dynamics
2. Sumith Y, Maroo SC. A direct two-dimensional pressure formulation in molecular dynamics. *J Mol Graph Model*. 2018;79:230–234.
3. Sumith Y, Maroo SC. A new algorithm for contact angle estimation in molecular dynamics simulations. *Int Conf Nanochannels Microchannels Minichannels*. 2015;
4. YD S, Maroo SC. A New Algorithm for Contact Angle Estimation in Molecular Dynamics Simulations. In: *ASME 2015 13th International Conference on Nanochannels, Microchannels, and Minichannels collocated with the ASME 2015 International Technical Conference and Exhibition on Packaging and Integration of Electronic and Photonic Microsystems*. American Society of Mechanical Engineers; 2015. p. V001T04A007–V001T04A007.
5. Daisy SY, Maroo SC. A robust algorithm for contact angle and interface detection of water and argon droplets. *Heat Transf Eng*. 2017;38(14–15):1343–1353.
6. Daisy SY. An efficient algorithm for contact angle estimation in molecular dynamics simulations. *Int J Eng*. 2015;9(1):1–8.
7. Yesudasan S, Averett RD. Coarse grain molecular dynamics simulation of fibrin polymerization. *ArXiv Prepr ArXiv171000123*. 2017;
8. Yesudasan S, Wang X, Averett RD. Coarse-grained molecular dynamics simulations of fibrin polymerization: effects of thrombin concentration on fibrin clot structure. *J Mol Model*. 2018;24(5):109.
9. Babu E, Yesudasan S, Chacko S. Cymatics Inspired Self-Cleaning Mechanism for Solar Panels. 2020;
10. Sumith Y, Chacko S. Dynamic analysis of a turbo-alternator shaft using finite element method. In: *International Conference on Advances in Mechanical Engineering (ICAME 2009)*. Universiti Teknologi Mara, Malaysia; 2009.
11. Yesudasan S. Extended MARTINI water model for heat transfer studies. *Mol Phys*. 2019;(0):1–9.
12. Sumith Y. Fast geometric fit algorithm for sphere using exact solution. 2015;
13. Yesudasan S. Fast geometric fit algorithm for sphere using exact solution. *ArXiv Prepr ArXiv150602776*. 2015;

14. Yesudasan S, Chacko S. Fast Local Pressure Estimation for Two Dimensional Systems From Molecular Dynamics Simulations. In: ASME Power Conference. American Society of Mechanical Engineers; 2018. p. V002T10A004.
15. Yesudasan S, Wang X, Averett RD. Fibrin polymerization simulation using a reactive dissipative particle dynamics method. *Biomech Model Mechanobiol*. 2018;17(5):1389–1403.
16. Averett RD, Yesudasan S, Scogin T, Walker ML. Finite Element Analysis of Magnetic Microparticle Induced Strain on a Fibrin Matrix due to the Influence of an Electromagnetic Field. *ArXiv Prepr ArXiv*. 2016;1611.
17. Hood JE, Yesudasan S, Averett RD. Glucose concentration affects fibrin clot structure and morphology as evidenced by fluorescence imaging and molecular simulations. *Clin Appl Thromb*. 2018;24(9_suppl):104S–116S.
18. Yesudasan S, Averett R, Chacko S. Machine Learned Coarse Grain Water Models for Evaporation Studies. *Preprints*. 2020;
19. Yesudasan S, Wang X, Averett RD. Molecular dynamics simulations indicate that deoxyhemoglobin, oxyhemoglobin, carboxyhemoglobin, and glycated hemoglobin under compression and shear exhibit an anisotropic mechanical behavior. *J Biomol Struct Dyn*. 2018;36(6):1417–1429.
20. Daisy SY. Molecular dynamics study of solid-liquid heat transfer and passive liquid flow [PhD Thesis]. Ph. D. Thesis, Syracuse University, Syracuse, USA; 2016.
21. Yesudasan Daisy S. Molecular Dynamics Study of Solid-Liquid Heat Transfer and Passive Liquid Flow. 2016;
22. Yesudasan S, Douglas SA, Platt MO, Wang X, Averett RD. Molecular insights into the irreversible mechanical behavior of sickle hemoglobin. *J Biomol Struct Dyn*. 2019;37(5):1270–1281.
23. Yesudasan S, Averett RD. Multiscale network model for fibrin fibers and fibrin clot with protofibril binding mechanics. *ArXiv Prepr ArXiv180804036*. 2018;
24. Yesudasan S, Averett RD. Multiscale Network Modeling of Fibrin Fibers and Fibrin Clots with Protofibril Binding Mechanics. *Polymers*. 2020;12(6):1223.
25. YD S, Maroo SC. Origin of Surface-Driven Passive Liquid Flows. *Langmuir*. 2016;
26. Yesudasan S, Averett RD. Recent advances in computational modeling of fibrin clot formation: a review. *Comput Biol Chem*. 2019;83:107148.
27. Noronha B, Yesudasan S, Chacko S. Static and Dynamic Analysis of Automotive Leaf Spring: A Comparative Study of Various Materials Using ANSYS.

MD code in Python

Yesudasan

28. YD S, Maroo SC. Surface-Heating Algorithm for Water at Nanoscale. *J Phys Chem Lett*. 2015;6(18):3765–3769.
29. Hotchandani V, Mathew B, Yesudasan S, Chacko S. Thermo-hydraulic characteristics of novel MEMS heat sink. *Microsyst Technol*. 2020;1–13.
30. Yesudasan S. Thin Film Pressure Estimation of Argon and Water using LAMMPS. *Int J Eng IJE*. 2019;12(1):1.
31. Shah H. How to write Molecular dynamics / Monte Carlo code for large molecules [Internet]. 2020. Available from: <https://www.researchgate.net/post/How-to-write-Molecular-dynamics-Monte-Carlo-code-for-large-molecules>
32. user3225087. periodic boundary conditions for triclinic box [Internet]. 2015. Available from: <https://scicomp.stackexchange.com/questions/20165/periodic-boundary-conditions-for-triclinic-box>
33. Yesudasan S. Python_MD [Internet]. 2020. Available from: https://github.com/ydsumith/Molecular-Dynamics/tree/master/Python_MD